CS8381 DATA STRUCTURES LABORATORY L T P C 0 0 4 2

OBJECTIVES

- To implement linear and non-linear data structures
- To understand the different operations of search trees
- To implement graph traversal algorithms
- To get familiarized to sorting and searching algorithms

Experiments:

1. Array implementation of Stack and Queue ADTs
2. Array implementation of List ADT
3. Linked list implementation of List, Stack and Queue ADTs
4. Applications of List, Stack and Queue ADTs
5. Implementation of Binary Trees and operations of Binary Trees
6. Implementation of Binary Search Trees
7. Implementation of AVL Trees
8. Implementation of Heaps using Priority Queues.
9. Graph representation and Traversal algorithms
10. Applications of Graphs
11. Implementation of searching and sorting algorithms
12. Hashing – any two collision techniques

TOTAL: 60 PERIODS

OUTCOMES: At the end of the course, the students will be able to:

- Write functions to implement linear and non-linear data structure operations
- Suggest appropriate linear / non-linear data structure operations for solving a given problem
- Appropriately use the linear / non-linear data structure operations for a given problem
- Apply appropriate hash functions that result in a collision free scenario for data storage and retrieval

EXP. NO : 1(a) **Implementation of Stack using Array**

AIM:

To write a 'C' program for the implementation of stack using array concept.

Algorithm:

```
1. Define a array which stores stack elements..
2. The operations on the stack are
        a)PUSH data into the stack
        b)POP data out of stack
3. PUSH DATA INTO STACK
        3a.  Enter the data to be inserted into stack.
        3b.  If TOP is NULL
                the input data is the first node in stack.
                the link of the node is NULL.
                TOP points to that node.
        3c.  If TOP is NOT NULL
                the link of TOP points to the new node.
                TOP points to that node.
4. POP DATA FROM STACK
        4a.If TOP is NULL
                the stack is empty
        4b.If TOP is NOT NULL
                the link of TOP is the current TOP.
                the pervious TOP is popped from stack.
5. The stack represented by linked list is traversed to display its content.
```

PROGRAM :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 100

void push(int x);
void pop();
void display();

int stk[MAX];
int top = -1;

void main()
{
int ch,data;
clrscr();

while(1)
{
printf("\n\n1.Push  2.Pop  3.Display  4.Exit\n\nEnter your choice  :");
scanf("%d",&ch);

switch(ch)
{
case 1:
printf("\nEnter the data   :");
scanf("%d",&data);
push(data);
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
exit(0);
}
}
getch();
}

void push(int x)
{
if(top > (MAX-1))
{
printf("\nStack over flow");
return;
}
stk[++top]=x;
printf("\nInserted : %d",x);
}

void pop()
{
if(top<0)
{
printf("\nStack under flow");
return;
}
printf("\nDeleted : %d",stk[top--]);
}

void display()
{
int i;
if(top<0)
{
printf("\nStack empty");
return;
}
for(i=top;i>=0;i--)
printf("%d \n",stk[i]);
}
```

OUTPUT:

```
1.Push   2.Pop   3.Display   4.Exit

Enter your choice   : 1

Enter the data   : 11
Inserted   : 11

1.Push   2.Pop   3.Display   4.Exit

Enter your choice   : 1

Enter the data   : 12
Inserted   : 12

1.Push   2.Pop   3.Display   4.Exit

Enter your choice   : 1

Enter the data   : 13
Inserted   : 13

1.Push   2.Pop   3.Display   4.Exit

Enter your choice   : 3

13
12
11

1.Push   2.Pop   3.Display   4.Exit

Enter your choice   : 2

Deleted  : 13

1.Push   2.Pop   3.Display   4.Exit

Enter your choice   : 3

12
11

1.Push   2.Pop   3.Display   4.Exit

Enter your choice   : 4
```

EXP. NO : 1(b) **Implementation of Queue ADT using an Array**

AIM:

To write a ‘c’ program for the implementation of Queue ADT using an Array.

ALGORITHM:

```
1. Define a array which stores queue elements..
2. The operations on the queue are
            a)INSERT data into the queue
            b)DELETE data out of queue
3. INSERT DATA INTO queue
      3a.Enter the data to be inserted into queue.
      3b.If TOP is NULL
            the input data is the first node in queue.
            the link of the node is NULL.
            TOP points to that node.
      3c.If TOP is NOT NULL
            the link of TOP points to the new node.
            TOP points to that node.
4. DELETE DATA FROM queue
      4a.If TOP is NULL
            the queue is empty
      4b.If TOP is NOT NULL
            the link of TOP is the current TOP.
            the pervious TOP is popped from queue.
5. The queue represented by linked list is traversed to display its content.
```

PROGRAM :

```
#include <stdio.h>
 #define MAX 50
int queue_array[MAX];
int rear = - 1;
int front = - 1;

void main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
             insert();
             break;
            case 2:
             delete();
             break;
            case 3:
             display();
             break;
            case 4:
```

```
            exit(1);
           default:
            printf("Wrong choice \n");
        }
    }
}
void insert()
{
    int add_item;
    if (rear == MAX - 1)
            printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
             front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
}
 void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
}
void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}
```

OUTPUT :

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1

Inset the element in queue : 10


1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1

Inset the element in queue : 15


1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1

Inset the element in queue : 20


1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1

Inset the element in queue : 30


1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2

Element deleted from queue is : 10


1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 3

Queue is :
15 20 30
```

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 4
```

RESULT :

Thus the program for the implementation of Stack and Queue ADT using an Array is executed and the output is verified.

EXP. NO : 2 **Implementation of List using Array**

AIM : To write a C program for the implementation of List using Array.

ALGORITHM:

```
1. Create nodes first, last, next, prev and cur then set the value as NULL.
2. Read the list operation type.
3. If operation type is create then process the following steps.
        i.   Allocate memory for node cur.
        ii.  Read data in cur's data area.
        iii. Assign cur node as NULL.
        iv.  Assign first=last=cur.
4. If operation type is Insert then process the following steps.
        i.   Allocate memory for node cur.
        ii.  Read data in cur's data area.
        iii. Read the position the Data to be insert.
        iv.  Availability of the position is true then assing cur's node as first and first=cur.
        v.   If availability of position is false then do following steps.
             1. Assign next as cur and count as zero.
             2. Repeat the following steps until count less than postion.
                     i.   Assign prev as next
                     ii.  Next as prev of node.
                     iii. Add count by one.
                     iv.  If prev as NULL then display the message INVALID POSITION.
                     v.   If prev not qual to NULL then do the following steps.
                               a. Assign cur's node as prev's node.
                               b. Assign prev's node as cur.
5. If operation type is delete then do the following steps.
        i.Read the position .
        ii.Check list is Empty .If it is true display the message List empty.
        iii.If position is first.
             1. Assign cur as first.
             2. Assign First as first of node.
             3. Reallocate the cur from memory.
             4. If position is last.
                     a. Move the current node to prev.
                     b. cur's node as Null.
                     c. Reallocate the Last from memory.
                     d. Assign last as cur.
                     e. If position is enter Mediate.
                               i.   Move the cur to required postion.
                               ii.  Move the Previous to cur's previous position
                               iii. Move the Next to cur's Next position.
                               iv.  Now Assign previous of node as next.
                               v.   Reallocate the cur from memory.
6. If operation is traverse.
     1. Assign current as first.
     2. Repeat the following steps until cur becomes NULL.
```

PROGRAM:

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

#define MAX 100
```

```
int list[MAX];
int element, pos, top=-1;

int menu(void);
void create(void);
void insert(int, int);
void delet(int);
void find(int);
void display(void);

void main()
{
int ch;

while(1)
{
ch = menu();
switch (ch)
{
case 1:
        top = 0;
        create();
        break;
case 2:
        if (top != MAX)
        {
        printf("“"\tEnter the New element : ");
        scanf("“%d”,&element);
        printf("“"\tEnter the Position : ");
        scanf("“%d”,&pos);
        insert(element, pos);
        }
        else
        {
        printf("“"\tList if Full. Cannot insert");
        printf("“"\nPress any key to continue...");
        getch();
        }
        break;

case 3:
        if (top != -1)
        {
        printf("“"Enter the position of element to be deleted : ");
        scanf("“%d”,&pos);
        delet(pos);
        }
        else
        {
        printf("“"List is Empty.");
        printf("“"\nPress any key to continue...");
        getch();
        }
        break;
case 4:
        printf("“"Enter the element to be searched : ");
        scanf("“%d”,&element);
        find(element);
        break;
```

```
case 5:
            display();
            break;
case 6:
            exit(0);
            break;
default:
            printf(“"Invalid Choice");
            printf(“"\nPress any key to continue...");
            getch();
}
}
}

int menu()
{
int ch;
clrscr();
printf(“"\n\t\t********************************************\n\n");
printf(“"\t\t******Implementation of List using array******\n\n");
printf(“"\t\t********************************************\n\n");
printf(“"\t1. Create\n\t2. Insert\n\t3. Delete\n\t4. Find\n\t5. Display\n\t6.Exit ");
printf(“"\n\n\tEnter your choice : ");
scanf(“%d”,&ch);
printf(“"\n\n");
return ch;
}

void create(void)
{
int flag=1;
while(flag==1 && top != MAX)
{
printf(“"Enter an element : ");
scanf(“%d”,&element);
list[top] = element;
top++;
printf(“"To insert another element press '1' : ");
scanf(“%d”,&flag);
}
}
void display(void)
{
int i;
for (i=0; i<top; i++)
printf(“"\nElement- %d :",i+1,list[i]);
printf(“"\n\nPress any key to continue...";
getch();
}

void insert(int element, int pos)
{
int i;
if (pos == 0)
{
            printf(“"\n\nCannot insert at zeroth position";
            getch();
            return;
}
```

```
if (pos-1 > top)
{
        printf(“"\n\nOnly %d elements exit. Cannot insert at %d position",top,pos);
        printf(“"\nPress any key to continue...";
        getch();
}
else
{
        for (i=top; i>=pos-1; i--)
                list[i+1] = list[i];
        list[pos-1] = element;
        top++;
}
}

void delet(int pos)
{
int i;
if(pos == 0)
{
        printf(“"\n\nCannot delete at zeroth position";
        getch();
        return;
}
if (pos > top)
{
        printf(“"\n\nOnly %d elements exit. Cannot delete",top);
        printf(“"\nPress any key to continue...";
        getch();
        return;
}
for (i=pos-1; i<top; i++)
        list[i] = list[i+1];
top--;
}


void find(int element)
{
int i;
int flag = 1;
for (i=0; i<top; i++)
{
        if(list[i] == element)
        {
                printf(“%d  exists at %d position",element,i+1);
                flag = 0;
                printf(“"\nPress any key to continue...";
                getch();
                break;
        }
}
if(flag == 1)
{
        printf(“"Element not found.\nPress any key to continue...";
        getch();
}
}
```

OUTPUT:

```
            ******************************************
             ****** Implementation of List using array********
            ******************************************

      1. Create
      2. Insert
      3. Delete
      4. Find
      5. Display
      6.Exit

      Enter your choice :1

Enter an element : 11
To insert another element press '1' : 1
Enter an element : 12
To insert another element press '1' : 1
Enter an element : 14
To insert another element press '1' : 2

      1. Create
      2. Insert
      3. Delete
      4. Find
      5. Display
      6.Exit

      Enter your choice : 5

Element-1 :11
Element-2 :12
Element-3 :14

Press any key to continue...

      1. Create
      2. Insert
      3. Delete
      4. Find
      5. Display
      6.Exit

      Enter your choice : 2


      Enter the New element : 13
      Enter the Position : 3

      1. Create
      2. Insert
      3. Delete
      4. Find
      5. Display
      6.Exit

      Enter your choice : 5
```

```
Element-1 :11
Element-2 :12
Element-3 :13
Element-4 :14

Press any key to continue...

    1. Create
    2. Insert
    3. Delete
    4. Find
    5. Display
    6.Exit

    Enter your choice : 3

Enter the position of element to be deleted : 4

    1. Create
    2. Insert
    3. Delete
    4. Find
    5. Display
    6.Exit

    Enter your choice : 5

Element-1 :11
Element-2 :12
Element-3 :13

Press any key to continue...

    1. Create
    2. Insert
    3. Delete
    4. Find
    5. Display
    6.Exit

    Enter your choice : 4

Enter the element to be searched : 12
12 exists at 2 position

Press any key to continue...

    1. Create
    2. Insert
    3. Delete
    4. Find
    5. Display
    6.Exit

    Enter your choice : 6
```

RESULT :

Thus the C program for the implementation of List using Array was performed and successfully verified.

EXP. NO : 3(a) **Implementation of List using Linked List**

AIM :

To write a C program for the implementation of List using Linked List.

ALGORITHM:

1. Allocate the memory address for the new node using malloc() function for inserting at first ,intermediate and at last.
2. To insert the new node at first. Read the data and check whether head is null or not.
3. If null then assign head to new node otherwise assign head to new node->next and new node to head.
4. To insert at intermediate, read the data and position in which it has to inserted.
5. If head is null, then assign it to head otherwise do the traversing process.
6. To insert at last then assign temp->next and null to new node->next it head is not null.
7. To delete the first element check if the head is null then print that list is empty.
8. Else assign head->next to head.
9. To delete the last element. Find the previous node of the last element and assign null to prev->next.
10. To delete the intermediate element find the previous and next node that has to be deleted.
11. Then assign temp->next to prev->next.
12. To display the list element that if head is null then print “list is empty”.
13. Else traverse the list from head to last node and during that print the node data.

PROGRAM:

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

struct node
{
	int no;
	struct node*next;
}
*newnode,*head=NULL,*temp,*prev;
typedef struct node s;

void main()
{
int t=0,x,cnt=0,pos=1,y,i=1,p,nod,e,v;

void display(s*);
int callcount(s*);
clrscr();

printf("Enter the no of node for creating the list\n");
scanf("%d",&nod);
while(i<=nod)
{
	newnode=(s*)malloc(sizeof(s));
	printf("Enter the data value %d for node\n",i);
	scanf("%d",&newnode->no);


	if(head==NULL)
	{
			head=newnode;
			head->next=NULL;
			temp=head;
		}
		else
		{
			temp->next=newnode;
			newnode->next=NULL;
			temp=newnode;
		}
		i++;
}
display(head);

L1:
printf("\n\t\t\tMain menu\n1.Insert first\t2.insert last\t3.Insert at given position\n");
printf("4.Delete First\t5.Delete last\t6.Delete at given position\n");
printf("7.Find the node\t8. Modify the node\t9.Count the number of node in List\n");
printf("Enter u'r choice\n");
scanf("%d",&y);

switch(y)
{
case  1:
		printf("\t\t\tU'r choice is insert first\n");
		newnode=(s*) malloc(sizeof(s));
		printf("Enter the element for node\n");
		scanf("%d",& newnode->no);
		if(head==NULL)
		{
			head=newnode;
			newnode->next=NULL;
		}
		else
		{
			newnode->next=head;
			head=newnode;
		}
		display(head);
		break;
case  2:
		printf("\t\t\tU'r choice is insert last\n");
		newnode=(s*)malloc (sizeof(s));
		printf("Enter the element for node\n");
		scanf("%d",& newnode->no);

		if(head==NULL)
		{
			head=newnode;
			newnode->next=NULL;
		}
		else
		{
			temp=head;
			while(temp->next!=NULL)
			{
				temp=temp->next;
			}
			temp->next=newnode;
```

```
                newnode->next=NULL;
        }
        display(head);
        break;
case  3:
        pos=1;
        printf("\t\t\tU'r choice is insert at given positiob\n");
        newnode=(s*) malloc(sizeof(s));
        printf("Enter the element for node\t");
        scanf("%d",& newnode->no);
        cnt=callcount(head);
        printf("Enter the position for insertion(between 1 to %d)\n",cnt+1);
        scanf("%d",&p);
        if(p<=(cnt+1))
        {
                temp=head;
                while(pos<p)
                {
                        prev=temp;
                        temp=temp->next;
                        pos++;
                }
                prev->next=newnode;
                newnode->next=temp;
                display(head);
        }
        else
                printf("Wrong position\n");
        break;
case  4:
        if(head==NULL)
                printf("The list is empty\n");
        else
        {
                printf("\t\t\t U'r choice is delete first\n");
                head=head->next;
                display(head);
        }
        break;

case  5:
        if(head==NULL)
                printf("The list is empty\n");
        else
        {
                printf("\t\t\tU'r choice is delete ladt\n");
                temp=head;
                while(temp->next!=NULL)
                {
                        prev=temp;
                        temp=temp->next;
                }
                prev->next=NULL;
                display(head);
        }
        break;

case  6:
        if(head==NULL)
```

```
                printf("The list is empty\n");
        else
        {
                pos=1;
                printf("\t\t\tU'r choice is delete at given position\n");
                cnt=callcount(head);
                printf("Enter the position for deletion (between 1 to %d)\n",cnt);
                scanf("%d",&p);
                if(p<=cnt)
                {
                        temp=head;
                        while(pos<p)
                        {
                                prev=temp;
                                temp=temp->next;
                                pos++;
                        }
                        prev->next=temp->next;
                        display(head);
                }
                else
                        printf("Wrong position\n");
        }
        break;
case  7:
        printf("\t\t\tU'r choice is find the node in the list\n");
        printf("Enter the data to be find\n");
        scanf("%d",&p);
        temp=head;
        t=0;
        while(temp->next!=NULL)
        {
                if(p==temp->no)
                {
                        t=1;
                        printf("The data %d is found\n",temp->no);
                }
                temp=temp->next;
        }

        if(p==temp->no)
        {
                t=1;
                printf("The data % d not found\n",p);
        }
        if(t==0)
                printf("The data %d not found\n",p);
        display(head);
        break;
case  8:
        printf("\t\t\tU'r choice is modify the node in the list\n");
        printf("Enter the data to be modified\n");
        scanf("%d",&p);
        temp=head;
        t=0;
        while(temp->next!=NULL)
        {
                if(temp->no==p)
                {
                        t=1;
                        printf("Enter the new data\n");
                        scanf("%d",&pos);
                        temp->no=pos;
                        display(head);
                        break;
                }
        temp=temp->next;
        }
        if(temp->no==p)
        {
                t=1;
                printf("Enter the new data\n");
                scanf("%d",&pos);
                temp->no=pos;
                display(head);
        }
        if(t==0)
                printf("The data '%d' not found is the list\n",p);
        break;

case   9:
        printf("\t\t\tU'r choice is count the number of node in the list\n");
        cnt=callcount(head);
        printf("Number of node in the list is... %d\n",cnt);
        display(head);
        break;
default:
        printf("\n Terminate the execution\n");
        break;
}


if((y>=1)&&(y<10))
{
        printf("\nPress 1 to display the main menu\n");
        scanf("%d",&v);
        if(v==1)
        goto L1;
}
getch();
}

void display(s*head)
{
s*temp=head;
if(temp==NULL)
        printf("The list is empty\n");
else
{
        printf("The list is...\n");
        while(temp->next!=NULL)
        {
                printf("%d-->",temp->no);
                temp=temp->next;
        }
        printf("%d",temp->no);
}
}
```

```
int callcount(s*head)
{
s*temp=head;
int cnt=0;
while(temp->next!=NULL)
{
        cnt++;
        temp=temp->next;
}
cnt++;
return cnt;
}
```

OUTPUT:-

```
Enter the no: of  node for creating the List
2
Enter the data value 1 for node
10
Enter the data value 2 for node
20
The List is . . .
10→20

                    Main menu
1.Insert First              2.Insert Last               3.Insert at given position
4.Delete First              5.Delete Last               6.Delete at given position
7.Find the node             8.Modify the node           9.Count the number of node in List
Enter u'r choice is Insert First
1

                    U'r choice is insert First
Enter the element for node
5
The List is. . . .
5→10→20
Presss 1 to display the Main menu
1

1.Insert First              2.Insert Last               3.Insert at given position
4.Delete Firt               5.Delete Last               6.Delete at given position
7.Find the node             8.Modify the nosde          9.Count the number of node in List
Enter u'r choice
2
                              U'r choice is Insert last

Enter the element for node
30
The List is . . .
5→10→20→30
Press 1 to display the Main menu
1

                              Main menu
1.Insert First              2.Insert Last               3.Insert at given position
4.Delete First              5.Delete Last               6.Delete at given position
7.Find the node             8.Modify the node           9.Count the number of node in List
Enter u'r choice
3
                    U'r Choice is Insert at given position

Enter the element for Insertion (between 1 to 5)
3
The  list is . . .
5→10→15→20→30
Press 1 to display the Main menu
1

                    Main menu
1.Insert First              2.Insert Last               3.Insert at given position
4.Delete First              5.Delete Last               6.Delete at given position
7.Find the node             8.Modify the node           9.Count the number of node in List
```

```
Enter u’r choice
4
                         U’r choice is Delete First
The List is . . .
10→15→20→30
Press 1 to display the Main menu
1

                         Main menu
1.Insert First                  2.Insert Last                   3.Insert at given position
4.Delete First                  5.Delete Last                   6.Delete at given position
7.find the node                 8.Modify the node               9.Count the number of node in List
Enter u’r choice
5
                         U’r choice is Delete Last
The List is. . .
10→15→20
Press 1 to display the Main menu
1

                         Main menu
1.Insert First                  2.Insert Last                   3.Insert at given position
4.Delete First                  5.Delete Last                   6.Delete at given position
7.find the node                 8.Modify the node               9.Count the number of node in List
Enter u’r choice
6
                         U’r choice is Delete at given position

Enter the position for Deletion (between 1 to 3)
2
The List is. . .
10→20
Press 1 to display the Main menu
1

                         Main menu
1.Insert First                  2.Insert Last                   3.Insert at given position
4.Delete First                  5.Delete Last                   6.Delete at given position
7.find the node                 8.Modify the node               9.Count the number of node in List
Enter u’r choice
7
                         U’r choice is find the node in the List
Enter the data to be Find

20
The data 20 is found
The List is. . .
10→20
Press 1 to display the Main menu
1
                         Main menu
1.Insert First                  2.Insert Last                   3.Insert at given position
4.Delete First                  5.Delete Last                   6.Delete at given position
7.find the node                 8.Modify the node               9.Count the number of node in List
Enter u’r choice
8
                         U’r Choice is Modify the node in the List
Enter the data to be modified
20
```

```
Enter the new data
15
The List is . .
10→15

Press 1 to display the main menu
1
                              Main menu
1.Insert First                2.Insert Last                 3.Insert at given position
4.Delete First                5.Delete Last                 6.Delete at given position
7.find the node               8.Modify the node             9.Count the number of node in List
Enter u'r choice
9
                              U'r choice is count the number of node in the List

Number of node in the List is . . . 2
The list is . .
10→15

Press 1 to display the Main menu
1
                              Main menu
1.Insert First                2.Insert Last                 3.Insert at given position
4.Delete First                5.Delete Last                 6.Delete at given position
7.find the node               8.Modify the node             9.Count the number of node in List
Enter u'r choice
10
```

EXP. NO : 3(b) **Implementation of Stack using Linked List**

AIM:

To write a ‘C’ program for the implementation of stack using Linked List.

ALGORITHM:

```
1. Define a array which stores stack elements..
2. The operations on the stack are
            a)PUSH data into the stack
            b)POP data out of stack
3. PUSH DATA INTO STACK
            3a.  Enter the data to be inserted into stack.
            3b.  If TOP is NULL
                    the input data is the first node in stack.
                    the link of the node is NULL.
                    TOP points to that node.
            3c.  If TOP is NOT NULL
                    the link of TOP points to the new node.
                    TOP points to that node.
4. POP DATA FROM STACK
            4a.If TOP is NULL
                    the stack is empty
            4b.If TOP is NOT NULL
                    the link of TOP is the current TOP.
                    the pervious TOP is popped from stack.
5. The stack represented by linked list is traversed to display its content.
```

PROGRAM :

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

struct Node
{
   int Data;
   struct Node *next;
}*head;

void popStack()
{
   struct Node *temp, *var=head;
   if(var==head)
   {
      head = head->next;
      free(var);
   }
   else
   printf("\nStack Empty");
}

void push(int value)
{
   struct Node *temp;
   temp=(struct Node *)malloc(sizeof(struct Node));
   temp->Data=value;
```

```
    if (head == NULL)
    {
         head=temp;
         head->next=NULL;
    }
    else
    {
        temp->next=head;
        head=temp;
    }
}

void display()
{
     struct Node *var=head;
     if(var!=NULL)
     {
          printf("\nElements are \n");
          while(var!=NULL)
          {
               printf("\t%d →",var->Data);
               var=var->next;
          }
     printf("\n");
     }
     else
     printf("\nStack is Empty");
}

int main(int argc, char *argv[])
{
     int i=0;
     head=NULL;
     printf(" \n1. Push to stack");
     printf(" \n2. Pop from Stack");
     printf(" \n3. Display data of Stack");
     printf(" \n4. Exit\n");
     while(1)
     {
          printf(" \nChoose Option: ");
          scanf("%d",&i);
          switch(i)
          {
               case 1:
               {
               int value;
               printf("\nEnter a value to push into Stack: ");
               scanf("%d",&value);
               push(value);
               display();
               break;
               }
               case 2:
               {
               popStack();
               display();
               break;
               }
               case 3:
```

```
            {
            display();
            break;
            }
            case 4:
            {
            struct Node *temp;
            while(head!=NULL)
            {
                temp = head->next;
                free(head);
                head=temp;
            }
            exit(0);
            }

            default:
            {
            printf("\nwrong choice for operation");
            }
        }
    }
}
```

OUTPUT :

```
1. Push to stack
2. Pop from Stack
3. Display data of Stack
4. Exit
Choose Option: 1
Enter a value to push into Stack :  10

Elements are
10 →

Choose Option: 1
Enter a value to push into Stack :  20

Elements are
20 → 10 →

Choose Option: 1
Enter a value to push into Stack :  30

Elements are
30 → 20 → 10

Choose Option: 1
Enter a value to push into Stack :  40

Elements are
10 → 20 → 30 → 40

Choose Option: 2

Elements are
10 → 20 → 30

Choose Option: 3
Elements are

10 → 20 → 30

Choose Option: 4
```

EXP. NO : 3(c) **Implementation of Queue ADT using Linked List**

AIM:

To write a 'C' program for the implementation of Queue ADT using Linked List.

ALGORITHM:

```
1. Define a struct for each node in the queue. Each node in the queue contains data and link to
      the next node. Front and rear pointer points to first and last node inserted in the queue.
2. The operations on the queue are
      2a.INSERT data into the queue
      2b.DELETE data out of queue
3. INSERT DATA INTO queue
      3a.Enter the data to be inserted into queue.
      3b.If TOP is NULL
            the input data is the first node in queue.
            the link of the node is NULL.
            TOP points to that node.
      3c.If TOP is NOT NULL
            the link of TOP points to the new node.
            TOP points to that node.
4. DELETE DATA FROM queue
      4a.If TOP is NULL
            the queue is empty
      4b.If TOP is NOT NULL
            the link of TOP is the current TOP.
            the pervious TOP is popped from queue.
5. The queue represented by linked list is traversed to display its content.
```

PROGRAM :

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

struct node
{
   int info;
   struct node *ptr;
}*front,*rear,*temp,*front1;

void enq(int data);
void deq();
void display();
void create();
int count = 0;


void main()
{
   int no, ch, e;
   clrscr();

   printf("\n 1 - Enqueue");
```

```
    printf("\n 2 - Dequeue");
    printf("\n 3 - Display");
    printf("\n 4 - Exit");
    create();

    while (1)
    {
            printf("\n Enter choice : ");
            scanf("%d", &ch);
            switch (ch)
            {
            case 1:
                printf("Enter data : ");
                scanf("%d", &no);
                enq(no);
                break;
            case 2:
                deq();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Wrong choice, Please enter correct choice  ");
                break;
            }
    }
}

void create()
{
    front = rear = NULL;
}
void enq(int data)
{
    if (rear == NULL)
    {
            rear = (struct node *)malloc(1*sizeof(struct node));
            rear->ptr = NULL;
            rear->info = data;
            front = rear;
    }
    else
    {
            temp=(struct node *)malloc(1*sizeof(struct node));
            rear->ptr = temp;
            temp->info = data;
            temp->ptr = NULL;
            rear = temp;
    }
    count++;
}

void display()
{
    front1 = front;
    if ((front1 == NULL) && (rear == NULL))
    {
            printf("Queue is empty");
            return;
    }
    while (front1 != rear)
    {
            printf("%d ", front1->info);
            front1 = front1->ptr;
    }
    if (front1 == rear)
            printf("%d", front1->info);
}

void deq()
{
    front1 = front;

    if (front1 == NULL)
    {
            printf("\n Error: Trying to display elements from empty queue");
            return;
    }
    else
            if (front1->ptr != NULL)
            {
                front1 = front1->ptr;
                printf("\n Dequeued value : %d", front->info);
                free(front);
                front = front1;
            }
            else
            {
                printf("\n Dequeued value : %d", front->info);
                free(front);
                front = NULL;
                rear = NULL;
            }
            count--;
}
```

OUTPUT :

```
1 - Enqueue
2 – Dequeue
3 - Display
4 – Exit

Enter choice : 1
Enter data : 14

Enter choice : 1
Enter data : 85

Enter choice : 1
Enter data : 38

Enter choice : 3
14      85     38
Enter choice : 2

Enter choice : 2
Dequeued value : 14

Enter choice : 3
85     38

Enter choice : 4
```

RESULT :

Thus the program for the implementation of List, Stack and Queue ADT using Linked list were executed and the output is verified.

EXP. NO : 4(a) **Addition of two Polynomial Expression using Linked List**

AIM:

To write a 'C' program for the evaluation of postfix expression using Stack.

ALGORITHM:

1. Let p and q be the two polynomials represented by linked lists
2. while p and q are not null, repeat step 2.
3. If powers of the two terms ate equal then if the terms do not cancel then insert the sum of the terms into the sum Polynomial Advance p Advance q Else if the power of the first polynomial> power of second Then insert the term from first polynomial into sum polynomial Advance p Else insert the term from second polynomial into sum polynomial Advance q
4. copy the remaining terms from the non empty polynomial into the sum polynomial. This step of the algorithm is to be processed till the end of the polynomials has not been reached.

PROGRAM :

```
#include <stdio.h>
typedef struct pnode
{
        float coef;
        int exp;
        struct pnode *next;
}p;

p *getnode();

void main()
{
p *p1,*p2,*p3;

p *getpoly(),*add(p*,p*);

void display(p*);
clrscr();
printf("\n Enter first polynomial");
p1=getpoly();
printf("\n Enter second polynomial");
p2=getpoly();
printf("\n The first polynomial is");
display(p1);
printf("\n The second polynomial is");
display(p2);
p3=add(p1,p2);
printf("\n Addition of two polynomial is :\n");
display(p3);
}

p *getpoly()
{
p *temp,*New,*last;
```

```
int flag,exp;
char ans;
float coef;
temp=NULL;
flag=1;
printf("\n Enter the polynomial in descending order of exponent");
do
{
        printf("\n Enter the coeff & exponent of a term");
        scanf("%f%d",&coef,&exp);
        New=getnode();
        if(New==NULL)
        printf("\n Memory cannot be allocated");
        New->coef=coef;
        New->exp=exp;
        if(flag==1)
        {
                temp=New;
                last=temp;
                flag=0;
        }
        else
        {
                last->next=New;
                last=New;
        }
        printf("\n Do you want to more terms");
        ans=getch();
}
while(ans=='y');
return(temp);
}

p *getnode()
{
p *temp;
temp=(p*) malloc (sizeof(p));
temp->next=NULL;
return(temp);
}

void display(p*head)
{
p*temp;
temp=head;
if(temp==NULL)
printf("\n Polynomial empty");
while(temp->next!=NULL)
{
        printf("%0.1fx^%d+",temp->coef,temp->exp);
        temp=temp->next;
}
printf("\n%0.1fx^%d",temp->coef,temp->exp);
getch();
}

p*add(p*first,p*second)
{
p *p1,*p2,*temp,*dummy;
char ch;
float coef;
p *append(int,float,p*);
p1=first;
p2=second;
temp=(p*)malloc(sizeof(p));
if(temp==NULL)
        printf("\n Memory cannot be allocated");
dummy=temp;
while(p1!=NULL&&p2!=NULL)
{
        if(p1->exp==p2->exp)
        {
                coef=p1->coef+p2->coef;
                temp=append(p1->exp,coef,temp);
                p1=p1->next;
                p2=p2->next;
        }
        else if(p1->exp<p2->exp)
        {
                coef=p2->coef;
                temp=append(p2->exp,coef,temp);
                p2=p2->next;
        }
        else if(p1->exp>p2->exp)
        {
                coef=p1->coef;
                temp=append(p1->exp,coef,temp);
                p1=p1->next;
        }
}
while(p1!=NULL)
{
        temp=append(p1->exp,p1->coef,temp);
        p1=p1->next;
}
while(p2!=NULL)
{
        temp=append(p2->exp,p2->coef,temp);
        p2=p2->next;
}
temp->next=NULL;
temp=dummy->next;
```

```
free(dummy);
return(temp);
}
p*append(int Exp,float Coef,p*temp)
{
p*New,*dum;
New=(p*)malloc(sizeof(p));
if(New==NULL)
        printf("\n Cannot be allocated");
New->exp=Exp;
New->coef=Coef;
New->next=NULL;
dum=temp;
dum->next=New;
dum=New;
return(dum);
}
```

OUTPUT:

```
Enter first polynomial
Enter the polynomial in descending order of exponent
Enter the coeff & exponent of a term
3 5

Do you want to more terms
Enter the coeff & exponent of a term
5
3

Do you want to more terms
Enter the coeff & exponent of a term
-1
0

Do you want to more terms
Enter second polynomial
Enter the polynomial in descending order of exponent
Enter the coeff & exponent of a term
4 5

Do you want to more terms
Enter the coeff & exponent of a term
5
2
```

```
Do you want to more terms

The first polynomial is3.0x^5+5.0x^3+
-1.0x^0

The second polynomial is4.0x^5+
5.0x^2

Addition of two polynomial is :
7.0x^5+5.0x^3+5.0x^2+-1.0x^0
```

EXP. NO : 4(b) Conversion of infix expression into postfix expression using Stack

AIM:

To write a ‘C’ program for the conversion of infix expression into postfix expression using Stack.

ALGORITHM:

1. Scan the Infix string from left to right.
2. Initialize an empty stack.
3. 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.
    a. 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.
    b. Repeat this step till all the characters are scanned.
4. If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.
5. Return the Postfix string.

PROGRAM :

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

#define MAX 10
typedef struct node
{
        char data;
        struct node*next;
}stack;

int push(char);
int pop(char*);
void intopost(char[],char[]);
int indexpriority(char p[][2],char data);
stack *topstack=NULL;
```

```
char ip[MAX][2]={{'(',MAX},{')',0},{'\0',0},{'+',1},{'-‘,1},{'*',2},{'/',2},{'%',3},{'^',3}};
char sp[MAX][2]={{'(',0},{')',-1},{'\0',0},{'+',1},{'-',1},{'*',2},{'/',2},{'%',3},{'^',3}};

void main()
{
            char instr[20],poststr[20];
            clrscr();
            printf("enter the infix expression");
            scanf("%s",instr);
            intopost(instr,poststr);
            printf("the postfix expression is :%s",poststr);
            getch();
}

int push(char value)
{
            stack *newnode;
            newnode=(stack*)malloc(sizeof(stack));
            if(newnode==NULL)
                        return -1;
            newnode->data=value;
            newnode->next=topstack;
            topstack=newnode;
            return 0;
}

int pop(char *value)
{
            stack *temp;
            if(topstack==NULL)
                        return -1;
            temp=topstack;
            topstack=topstack->next;
            *value=temp->data;
            free(temp);
            return 0;
}

void intopost(char instr[],char poststr[])
{
char ch,item;
int i=0,st=0,spr,ipr;
push('\0');
while((ch=instr[st++])!=NULL)
{
            if(tolower(ch)>='a'&&tolower(ch)<='z')
                        poststr[i++]=ch;
            else if(ch==')')
            {
                        pop(&item);
                        while(item!='(')
                        {
                        poststr[i++]=item;
                        pop(&item);
                        }
            }
            else
            {
                        pop(&item);
```

```
                spr=indexpriority(sp,item);
                ipr=indexpriority(ip,ch);



                while(sp[spr][1]>=ip[ipr][1])
                {
                poststr[i++]=item;
                pop(&item);
                spr=indexpriority(sp,item);
                }
                push(item);
                push(ch);
        }
}



while(!pop(&item))
        poststr[i++]=item;
}

int indexpriority(char p[][2],char data)
{
        int index;
        for(index=0;index<MAX;index++)
        if(p[index][0]==data)
        return index;
}
```

OUTPUT :

```
        Enter the infix expression a+b*c+d

        The postfix expression is: abc*+d+
```

EXP. NO : 4(c) **Evaluation of postfix expression using Stack**

AIM:

To write a ‘C’ program for the evaluation of postfix expression using Stack.

ALGORITHM:

1. Traverse from left to right of the expression.
2. If an operand is encountered, push it onto the stack.
3. If you see a binary operator, pop two elements from the stack, evaluate those operands with that operator, and push the result back in the stack.
4. If you see a unary operator, pop one elements from the stack, evaluate those operands with that operator, and push the result back in the stack.
5. When the evaluation of the entire expression is over, the only thing left on the stack should be the final result. If there are zero or more than 1 operands left on the stack, either your program is inconsistent, or the expression was invalid.

PROGRAM :

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

int top = -1;
int stack[100];

void push (int data)
{
stack[++top] = data;
}

int pop ()
{
int data;
if (top == -1)
   return -1;
data = stack[top];
stack[top] = 0;
top--;
return (data);
}
void main()
{
char str[100];
int i, data = -1, operand1, operand2, result;
clrscr();
printf("Enter your postfix expression:");
fgets(str, 100, stdin);
for (i = 0; i < strlen(str); i++)
{
            if (isdigit(str[i]))
            {
            data = (data == -1) ? 0 : data;
            data = (data * 10) + (str[i] - 48);

            continue;
            }
            if (data != -1)
            {
                  push(data);
            }
            if (str[i] == '+' || str[i] == '-'  || str[i] == '*' || str[i] == '/')
            {
                   operand2 = pop();
                   operand1 = pop();
            if (operand1 == -1 || operand2 == -1)
                   break;
            switch (str[i])
            {
            case '+':
                        result = operand1 + operand2;
                        push(result);
                        break;
            case '-':
                        result = operand1 - operand2;
```

```
				push(result);
				break;
				case '*':
				result = operand1 * operand2;
				push(result);
				break;
		case '/':
				result = operand1 / operand2;
				push(result);
				break;
		}
		}
		data = -1;
}
if (top == 0)
	printf("Output:%d\n", stack[top]);
else
	printf("You have given wrong postfix expression\n");
getch();
}
```

OUTPUT :

```
		Enter your postfix expression:   10  20  *  30  40  10  / - +

		Output :   226
```

RESULT :

Thus the program for the application of List, Stack and Queue ADT’s were executed and the output is verified.

EXP. NO : 5 **Implementation of Binary Trees and operations of Binary Trees**

AIM:

To write a ‘C’ program to implement the binary tree and its operations.

ALGORITHM:

1. If the tree is empty, initialize the root with new node.
2. Else, get the front node of the queue.
   a) If the left child of this front node doesn’t exist, set the left child as the new node.
   b) else if the right child of this front node doesn’t exist, set the right child as the new node.
3. If the front node has both the left child and right child, remove it.
4. Insert the new node.

PROGRAM :

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

struct bin_tree
{
int data;
struct bin_tree * right, * left;
};
typedef struct bin_tree node;

void insert(node ** tree, int val)
{
 node *temp = NULL;
 if(!(*tree))
{
        temp = (node *)malloc( sizeof(node));
        temp->left = temp->right = NULL;
        temp->data = val;
        *tree = temp;
        return;
 }
if(val < (*tree)->data)
{
        insert(&(*tree)->left, val);
}
else if(val > (*tree)->data)
{
        insert(&(*tree)->right, val);
}
}

void print_preorder(node * tree)
{
    if (tree)
    {
        printf("%d\n",tree->data);
        print_preorder(tree->left);
        print_preorder(tree->right);
    }

}
```

```
void print_inorder(node * tree)
{
if (tree)
{
        print_inorder(tree->left);
        printf("%d\n",tree->data);
        print_inorder(tree->right);
}
}
void print_postorder(node * tree)
{
if (tree)
{
        print_postorder(tree->left);
        print_postorder(tree->right);
        printf("%d\n",tree->data);
}
}
void deltree(node * tree)
{
if (tree)
{
        deltree(tree->left);
        deltree(tree->right);
        free(tree);
 }
}
node* search(node ** tree, int val)
{
 if(!(*tree))
{
        return NULL;
}
if(val < (*tree)->data)
{
        search(&((*tree)->left), val);
}
 else if(val > (*tree)->data)
{
        search(&((*tree)->right), val);
}
else if(val == (*tree)->data)
{
        return *tree;
}
}
void main()
{
    node *root;
    node *tmp;

    root = NULL;
    /* Inserting nodes into tree */
    insert(&root, 9);
    insert(&root, 4);
    insert(&root, 15);
    insert(&root, 6);
    insert(&root, 12);
    insert(&root, 17);
    insert(&root, 2);

    /* Printing nodes of tree */
    printf("Pre Order Display\n");
    print_preorder(root);

    printf("In Order Display\n");
    print_inorder(root);

    printf("Post Order Display\n");
    print_postorder(root);

    /* Search node into tree */
    tmp = search(&root, 4);
    if (tmp)
    {
        printf("Searched node=%d\n", tmp->data);
    }
    else
    {
        printf("Data Not found in tree.\n");
    }

    /* Deleting all nodes of tree */
    deltree(root);
}
```

OUTPUT :

```
Pre Order Display
9
4
2
6
15
12
17
In Order Display
2
4
6
9
12
15
17
Post Order Display
2
6
4
12
17
15
9
Searched node=4
```

RESULT :

Thus the program to implement the binary tree and its operations is executed and the output is verified.

EXP. NO : 6 **Implementation of Binary Search Trees**

AIM:

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

ALGORITHM:

1. Declare function create(),search(),delete(),Display().
2. Create a structure for a tree contains left pointer and right pointer.
3. Insert an element is by checking the top node and the leaf node and the operation will be performed.
4. Deleting an element contains searching the tree and deleting the item.
5. Display the Tree elements.

PROGRAM :

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

typedef struct BST
{
  int data;
  struct BST *lchild, *rchild;
} node;

void insert(node *, node *);
void inorder(node *);
void preorder(node *);
void postorder(node *);


void main()
{
  int choice;
  char ans = 'N';
  int key;
  node *new_node, *root, *tmp, *parent;
  node *get_node();
  root = NULL;
  clrscr();

  printf("\nProgram For Binary Search Tree ");
  do
  {
    printf("\n1.Create");
    printf("\n2.Recursive Traversals");
    printf("\n3.Exit");
    printf("\nEnter your choice :");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        do
        {
        new_node = get_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("\nWant To enter More Elements?(y/n)");
        ans = getch();
        } while (ans == 'y');
        break;

    case 2:
      if (root == NULL)
        printf("Tree Is Not Created");
      else
      {
        printf("\nThe Inorder display : ");
        inorder(root);
        printf("\nThe Preorder display : ");
        preorder(root);
        printf("\nThe Postorder display : ");
        postorder(root);
      }
      break;

    case 3:
           break;
    }
  } while (choice != 4);
}

/*  Get new Node  */
node *get_node()
{
  node *temp;
  temp = (node *) malloc(sizeof(node));
  temp->lchild = NULL;
  temp->rchild = NULL;
  return temp;
}

/*  This function is for creating a binary search tree  */
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);
 }
}
```

```
/* This function displays the tree in inorder fashion */
void inorder(node *temp)
{
 if (temp != NULL)
{
      inorder(temp->lchild);
      printf("%d  ", temp->data);
      inorder(temp->rchild);
 }
}

/*  This function displays the tree in preorder fashion  */
void preorder(node *temp)
{
if (temp != NULL)
{
      printf("%d   ", temp->data);
      preorder(temp->lchild);
      preorder(temp->rchild);
 }
}

/*  This function displays the tree in postorder fashion  */
void postorder(node *temp)
{
if (temp != NULL)
{
      postorder(temp->lchild);
      postorder(temp->rchild);
      printf("%d   ", temp->data);
 }
}
```

OUTPUT :

```
Program for Binary Search Tree
1.Create
2.Recursive Traversals
3.Exit
Enter your choice :1

Enter The Element 44

Want To enter More Elements?(y/n)
Enter The Element
33

Want To enter More Elements?(y/n)
Enter The Element
77

Want To enter More Elements?(y/n)
Enter The Element 22

Want To enter More Elements?(y/n)
1.Create
2.Recursive Traversals
3.Exit
Enter your choice :2

The Inorder display : 22 33 44 77
The Preorder display : 44 33 22 77
The  Postorder display : 22 33 77 44

1.Create
2.Recursive Traversals
3.Exit
Enter your choice :3
```

RESULT :

Thus the program to implement the binary search tree is executed and the output is verified.

EXP. NO : 7 **Implementation of AVL Trees**

AIM:

To write a 'C' program to implement the AVL tree.

ALGORITHM:

1. Insert the new Node using recursion so while back tracking you will all the parents nodes to check whether they are still balanced or not.
2. Every node has a field called height with default value as 1.
3. When new node is added, its parent's node height get increased by 1.
4. So as mentioned in step 1, every ancestors height will get updated while back tracking to the root.
5. At every node the balance factor will also be checked. balance factor = (height of left Subtree – height of right Subtree).
6. If balance factor =1 means tree is balanced at that node.
7. If balance factor >1 means tree is not balanced at that node, left height is more that the right height so that means we need rotation. (Either Left-Left Case or Left-Right Case).
8. Say the current node which we are checking is X and If new node is less than the X.left then it will be Left-Left case, and if new node is greater than the X.left then it will be Left-Right case. see the pictures above.
9. If balance factor <-1 means tree is not balanced at that node, right height is more that the left height so that means we need rotation. (Either Right-Right Case or Right- Left Case)
10. Say the current node which we are checking is X and If new node is less than the X.right then it will be Right-Right case, and if new node is greater than the X.right then it will be Right-Left case. see the pictures above

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");
                    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 program to implement the AVL tree is executed and the output is verified.

EXP. NO : 8 **Implementation of Heaps using Priority Queues**

AIM:

To write a ‘C’ program to implement the Heap using Priority Queue.

ALGORITHM:

1. Insert an item with priority in the priority queue. Create the priority queue by inserting more items by satisfying heap order property and structure property.
2. To delete an element from the priority queue (ie.,max-heap), delete the element from the root which is a maximum value. Then satisfy the heap order property and structure property.
3. Displaying the elements are also done by considering the priority (ie., maximum).

PROGRAM :

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

#define MAX 5

void insert_by_priority(int);
void delete_by_priority(int);
void create();
void check(int);
void display_pqueue();

int pri_que[MAX];
int front, rear;

void main()
{
int n, ch;

printf("\n1 - Insert an element into queue");
printf("\n2 - Delete an element from queue");
printf("\n3 - Display queue elements");
printf("\n4 - Exit");

create();

while (1)
{
printf("\nEnter your choice : ");
scanf("%d", &ch);

switch (ch)
{
case 1:
    printf("\nEnter value to be inserted : ");
    scanf("%d",&n);
    insert_by_priority(n);
    break;

case 2:
    printf("\nEnter value to delete : ");
    scanf("%d",&n);
```

```
        delete_by_priority(n);
        break;

case 3:
        display_pqueue();
        break;

case 4:
            exit(0);
default:
        printf("\nChoice is incorrect, Enter a correct choice");
}
}
}

/* Function to create an empty priority queue */
void create()
{
front = rear = -1;
}

/* Function to insert value into priority queue */
void insert_by_priority(int data)
{
if (rear >= MAX - 1)
{
printf("\nQueue overflow no more elements can be inserted");
return;
}
if ((front == -1) && (rear == -1))
{
front++;
rear++;
pri_que[rear] = data;
return;
}
else
check(data);
rear++;
}

/* Function to check priority and place element */
void check(int data)
{
int i,j;

for (i = 0; i <= rear; i++)
{
if (data >= pri_que[i])
{
for (j = rear + 1; j > i; j--)
{
                        pri_que[j] = pri_que[j - 1];
}
pri_que[i] = data;
return;
}
}
pri_que[i] = data;
```

```
}

/* Function to delete an element from queue */
void delete_by_priority(int data)
{
int i;

if ((front==-1) && (rear==-1))
{
printf("\nQueue is empty no elements to delete");
return;
}

for (i = 0; i <= rear; i++)
{
if (data == pri_que[i])
{
for (; i < rear; i++)
        pri_que[i] = pri_que[i + 1];
pri_que[i] = -99;
rear--;
if (rear == -1)
front = -1;
return;
}
}
printf("\n%d not found in queue to delete", data);
}

/* Function to display queue elements */
void display_pqueue()
{
if ((front == -1) && (rear == -1))
{
printf("\nQueue is empty");
return;
}

for (; front <= rear; front++)
        printf(" %d ", pri_que[front]);

front = 0;
}
```

OUTPUT:

```
1 - Insert an element into queue
2 - Delete an element from queue
3 - Display queue elements
4 - Exit
Enter your choice : 1


Enter value to be inserted : 20


Enter your choice : 1

Enter value to be inserted : 45


Enter your choice : 1

Enter value to be inserted : 89


Enter your choice : 3
 89  45  20


Enter your choice : 1

Enter value to be inserted : 56


Enter your choice : 3
 89  56  45  20

Enter your choice : 2

Enter value to delete : 45


Enter your choice : 3
 89  56  20


Enter your choice : 4
```

RESULT :

Thus the program to implement the Heap using Priority Queue is executed and the output is verified.

EXP. NO : 9(a) **Graph representation using Adjacency Matrix**

AIM:

To write a ‘C’ program to implement the Graph representation using Adjacency Matrix.

ALGORITHM:

1. In undirected graph, for each edge which is present between the vertices Vi and Vj is represented by using a pair of round vertices (Vi,Vj), then set the value ‘1’ in the matrix at (i,j) and (j,i). Else set the value ‘0’ in the matrix at (i,j).
2. In directed graph, if Vi and Vj nodes having an edge. then it is represented by a value ‘1’ in the matrix only at (i,j). Otherwise represent the value 0.

PROGRAM :

```
#include<stdio.h>
#define MAX 100

int adj[MAX][MAX]; /*Adjacency matrix*/
int n;    /*Number of vertices in the graph*/

int main()
{
        int max_edges,i,j,origin,destin;
        int graph_type;

        printf("\nEnter 1 for Undirected graph\nEnter 2 for Directed graph\n");
        printf("\nEnter your choice :: ");
        scanf("%d",&graph_type);

        printf("\nEnter number of vertices :: ");
        scanf("%d",&n);

        if(graph_type==1)
                max_edges = n*(n-1)/2;
        else
                max_edges = n*(n-1);

        for(i=1; i<=max_edges; i++)
        {
                printf("\nEnter edge [ %d ] ( -1 -1 to quit ) : ",i);
                scanf("%d %d",&origin,&destin);
                if( (origin == -1) && (destin == -1) )
                        break;
                if( origin>=n || destin>=n || origin<0 || destin<0 )
                {
                        printf("\nInvalid vertex!\n");
                        i--;
                }
                else
                {
                        adj[origin][destin] = 1;
                        if( graph_type == 1) /*if undirected graph*/
                                adj[destin][origin] = 1;
                }
        }/*End of for*/

        printf("\nThe adjacency matrix is :: \n");
        for(i=0; i<=n-1; i++)
        {
                for(j=0; j<=n-1; j++)
                        printf("%4d",adj[i][j]);
                printf("\n");
        }
}
```

OUTPUT :

```
Enter 1 for Undirected graph
Enter 2 for Directed graph

Enter your choice :: 2

Enter number of vertices :: 5

Enter edge [ 1 ] ( -1 -1 to quit ) : 0 1

Enter edge [ 2 ] ( -1 -1 to quit ) : 0 2

Enter edge [ 3 ] ( -1 -1 to quit ) : 0 3

Enter edge [ 4 ] ( -1 -1 to quit ) : 1 3

Enter edge [ 5 ] ( -1 -1 to quit ) : 2 3

Enter edge [ 6 ] ( -1 -1 to quit ) : 3 4

Enter edge [ 7 ] ( -1 -1 to quit ) : 4 2

Enter edge [ 8 ] ( -1 -1 to quit ) : -1 -1

The adjacency matrix is ::
   0   1   1   1   0

   0   0   0   1   0

   0   0   0   1   0

   0   0   0   0   1

   0   0   1   0   0


Process returned 0
```

EXP. NO : 9(b) **Graph representation using Adjacency List**

AIM:

To write a 'C' program to implement the Graph representation using Adjacency List.

ALGORITHM:

1. In undirected graph, for each edge which is present between the vertices Vi and Vj is represented by thr vertices (Vi,Vj), then create the node with data Vj from the head Vi and create the node with data Vi from the head Vj.
2. In directed graph, if Vi and Vj nodes having an edge. then create the node with data Vj from the head Vi.

PROGRAM :

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

struct node
{
   int vertex;
   struct node* next;
};
struct node* createNode(int);

struct Graph
{
   int numVertices;
   struct node** adjLists;
};

struct Graph* createGraph(int vertices);
void addEdge(struct Graph* graph, int src, int dest);
void printGraph(struct Graph* graph);

int main()
{
   struct Graph* graph = createGraph(6);
   addEdge(graph, 0, 1);
   addEdge(graph, 0, 2);
   addEdge(graph, 1, 2);
   addEdge(graph, 1, 4);
   addEdge(graph, 1, 3);
   addEdge(graph, 2, 4);
   addEdge(graph, 3, 4);
   addEdge(graph, 4, 6);
   addEdge(graph, 5, 1);
   addEdge(graph, 5, 6);

   printGraph(graph);
   return 0;
}
```

```
struct node* createNode(int v)
{
   struct node* newNode = malloc(sizeof(struct node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}

struct Graph* createGraph(int vertices)
{
   int i;
   struct Graph* graph = malloc(sizeof(struct Graph));
   graph->numVertices = vertices;

   graph->adjLists = malloc(vertices * sizeof(struct node*));

   for (i = 0; i < vertices; i++)
      graph->adjLists[i] = NULL;

   return graph;
}

void addEdge(struct Graph* graph, int src, int dest)
{
   // Add edge from src to dest
   struct node* newNode = createNode(dest);
   newNode->next = graph->adjLists[src];
   graph->adjLists[src] = newNode;

   // Add edge from dest to src
   newNode = createNode(src);
   newNode->next = graph->adjLists[dest];
   graph->adjLists[dest] = newNode;
}

void printGraph(struct Graph* graph)
{
   int v;
   for (v = 0; v < graph->numVertices; v++)
   {
      struct node* temp = graph->adjLists[v];
      printf("\n Adjacency list of vertex %d\n ", v);
      while (temp)
      {
         printf("%d -> ", temp->vertex);
         temp = temp->next;
      }
      printf("\n");
   }
}
```

OUTPUT :

```
Adjacency list of vertex 0
2 -> 1 ->

Adjacency list of vertex 1
5 -> 3 -> 4 -> 2 -> 0 ->

Adjacency list of vertex 2
4 -> 1 -> 0 ->

Adjacency list of vertex 3
4 -> 1 ->

Adjacency list of vertex 4
6 -> 3 -> 2 -> 1 ->

Adjacency list of vertex 5
6 -> 1 ->
```

EXP. NO : 9(c) **Graph Traversal using Depth First Search Algorithm**

AIM:

To write a ‘C’ program to implement the Graph Traversal using Depth First Search Algorithm.

ALGORITHM:

1. Pick a starting node and push all its adjacent nodes into a stack.
2. Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
3. Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

PROGRAM :

```
#include<stdio.h>

void DFS(int);
int G[10][10],visited[10],n;    //n is no of vertices and graph is sorted in array G[10][10]
void main()
{
   int i,j;
   printf("Enter number of vertices:");

   scanf("%d",&n);
   //read the adjecency matrix
   printf("\nEnter adjecency matrix of the graph:");

   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
           scanf("%d",&G[i][j]);

   //visited is initialized to zero
  for(i=0;i<n;i++)
       visited[i]=0;
   DFS(0);
}
void DFS(int i)
{
   int j;
   printf("\n%d",i);
   visited[i]=1;

   for(j=0;j<n;j++)
      if(!visited[j]&&G[i][j]==1)
           DFS(j);
}
```

OUTPUT :

```
Enter number of vertices:4

Enter adjacency matrix of the graph:

 0     1     1     1

1     0     1     0

1     1     0     1

1     0     1     0


0

1

2

3
```

EXP. NO : 9(d) **Graph Traversal using Breadth First Search Algorithm**

AIM:

To write a ‘C’ program to implement the Graph Traversal using Breadth First Search Algorithm.

ALGORITHM:

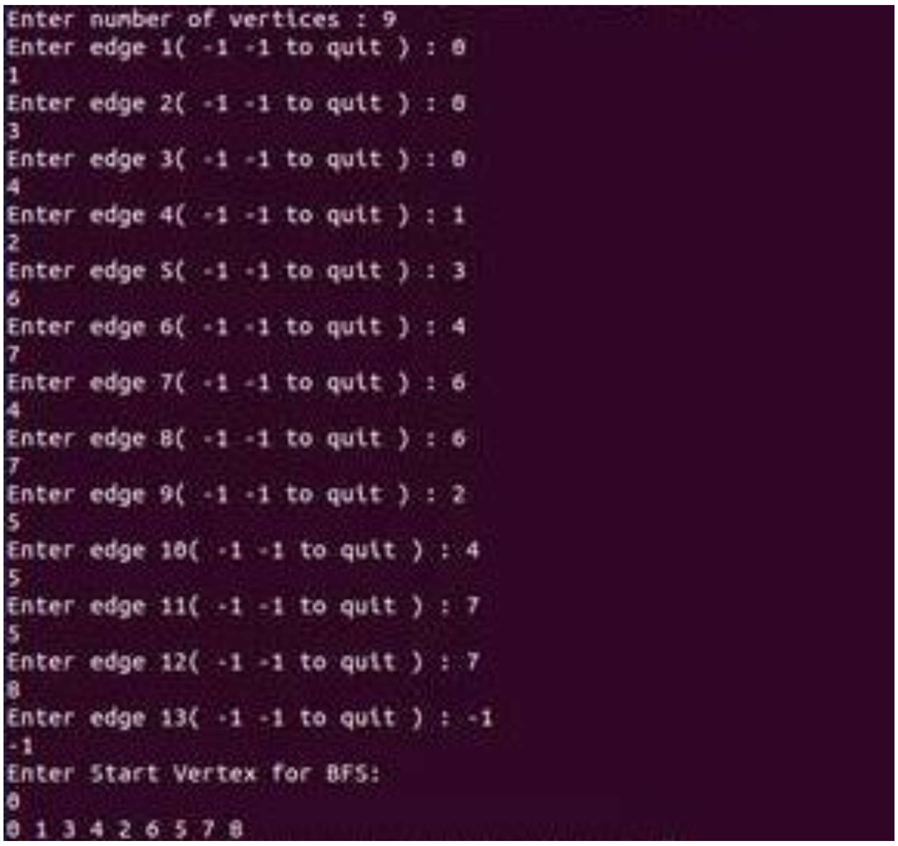
1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

PROGRAM :

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
   create_graph();
   BF_Traversal();
   return 0;
}

void BF_Traversal()
{
   int v;
   for(v=0; v<n; v++)
      state[v] = initial;

   printf("Enter Start Vertex for BFS: \n");
   scanf("%d", &v);
   BFS(v);
}
```

```
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



```
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
```

RESULT :

Thus the program to implement the Graph representation and Traversal algorithms were executed and the output is verified.

EXP. NO : 10 **Implementation of Graph Application for Travelling Salesman Problem**

AIM:

To write a ‘c’ program to implement the Graph Application for Travelling Salesman Problem.

ALGORITHM:

1. Consider city 1 as the starting and ending point.
2. Generate all (n-1)! Permutations of cities.
3. Calculate cost of every permutation and keep track of minimum cost permutation.
4. Return the permutation with minimum cost.

PROGRAM :

```
#include<stdio.h>
int ary[10][10],completed[10],n,cost=0;

void takeInput()
{
int i,j;
printf("Enter the number of villages: ");
scanf("%d",&n);
printf("\nEnter the Cost Matrix\n");
for(i=0;i < n;i++)
{
	printf("\nEnter Elements of Row: %d\n",i+1);
	for( j=0;j < n;j++)
		scanf("%d",&ary[i][j]);
	completed[i]=0;
}
printf("\n\nThe cost list is:");
for( i=0;i < n;i++)
{
	printf("\n");
	for(j=0;j < n;j++)
		printf("\t%d",ary[i][j]);
}
}

void mincost(int city)
{
int i,ncity;
completed[city]=1;
printf("%d--->",city+1);
ncity=least(city);
if(ncity==999)
{
	ncity=0;
	printf("%d",ncity+1);
	cost+=ary[city][ncity];
	return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i < n;i++)
{
        if((ary[c][i]!=0)&&(completed[i]==0))
                if(ary[c][i]+ary[i][c] < min)
                {
                        min=ary[i][0]+ary[c][i];
                        kmin=ary[c][i];
                        nc=i;
                }
}
if(min!=999)
        cost+=kmin;
return nc;
}

int main()
{
takeInput();
printf("\n\nThe Path is:\n");
mincost(0); //passing 0 because starting vertex
printf("\n\nMinimum cost is %d\n ",cost);
return 0;
}
```

OUTPUT:

```
Enter the number of villages: 4

Enter the Cost Matrix
Enter Elements of Row: 1
0 4 1 3
Enter Elements of Row: 2
4 0 2 1
Enter Elements of Row: 3
1 2 0 5
Enter Elements of Row: 4
3 1 5 0

The cost list is:
0 4 1 3
4 0 2 1
1 2 0 5
3 1 5 0

The Path is:
1—>3—>2—>4—>1

Minimum cost is 7
```

RESULT :

Thus the program to implement the Graph Application for Travelling Salesman Problem is executed and the output is verified.

EXP. NO : 11(a) **Implementation of searching and sorting algorithms – Insertion Sort**

AIM:

To write a ‘C’ program for the implementation of insertion sort.

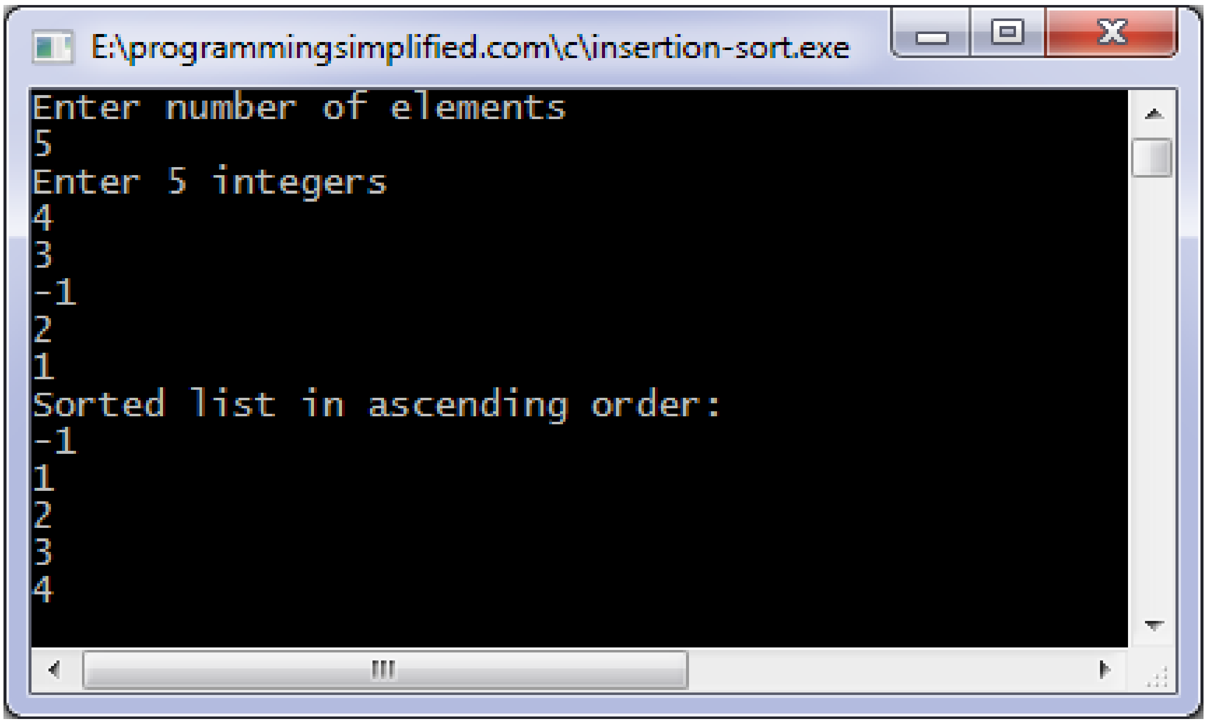
ALGORITHM :

1. Start with the result as the first element of the input.
2. Loop over the input until it is empty, "removing" the first remaining (leftmost) element.
3. Compare the removed element against the current result, starting from the highest
   a. (rightmost) element, and working left towards the lowest element.
4. If the removed input element is lower than the current result element, copy that value into
   a. the following element to make room for the new element below, and repeat with the next
   b. lowest result element.
5. Otherwise, the new element is in the correct location; save it in the cell left by copying the last examined result up, and start again from (2) with the next input element.

PROGRAM :

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

void main()
{
int n, array[1000], c, d, t;
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 = 1 ; c <= n - 1; c++)
{
        d = c;
        while ( d > 0 && array[d] < array[d-1])
        {
                t = array[d];
                array[d]   = array[d-1];
                array[d-1] = t;
                d--;
        }
}
printf("Sorted list in ascending order:\n");
for (c = 0; c <= n - 1; c++)
        printf("%d\n", array[c]);
getch();
}
```

OUTPUT :



```
E:\programmingsimplified.com\c\insertion-sort.exe
Enter number of elements
5
Enter 5 integers
4
3
-1
2
1
Sorted list in ascending order:
-1
1
2
3
4
```

EXP. NO : 11(b) **Implementation of searching and sorting algorithms – Shell Sort**

AIM:

To write a 'C' program for the implementation of shell sort.

ALGORITHM :

1. Starts by comparing elements far apart, then elements less far apart, and finally comparing adjacent elements.
2. In each step, the sortedness of the sequence is increased, until in the last step it is completely sorted. However, the number of sorting operations necessary in each step is limited, due to the presortedness of the sequence obtained in the preceding steps.
   - 2a. At first elements at distance $\mathbf{h_t}$ are sorted, then elements at distance $\mathbf{h_{t-1}}$ are sorted, etc, until finally the array is sorted using insertion sort (distance $h_1 = 1$).
   - 2b. An array is said to be $\mathbf{h_k}$-sorted if all elements spaced a distance $\mathbf{h_k}$ apart are sorted relative to each other.
   - 2c. Shell sort only works because an array that is $\mathbf{h_k}$-sorted remains $\mathbf{h_k}$-sorted when $\mathbf{h_{k-1}}$-sorted.

PROGRAM :

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

```
void main()
{
int array[5]={4,5,2,3,6},i1=0;

ShellSort(array,5);

printf("After Sorting:");

for(i1=0;i1<5;i1++)
        printf("%d",array[i1]);
 getch();
}

void ShellSort(int *array, int number_of_elements)
{
 int i, j, k, temp;
 for(k = number_of_elements/2;k > 0; k /= 2)
 {
         for(i = k; i<number_of_elements; i++)
         {
                 temp = array[i];

                 for(j = i; j >= k ;j-=k)
                 {
                         if(temp < array[j-k])
                                 array[j] = array[j-k];
                         else
                                 break;
                 }
                 array[j] = temp;
         }
 }
}
```

OUTPUT :

```
After Sorting :      2     3     4     5     6
```

EXP. NO : 11(c) **Implementation of searching and sorting algorithms – Selection Sort**

AIM:

To write a ‘C’ program for the implementation of selection sort.

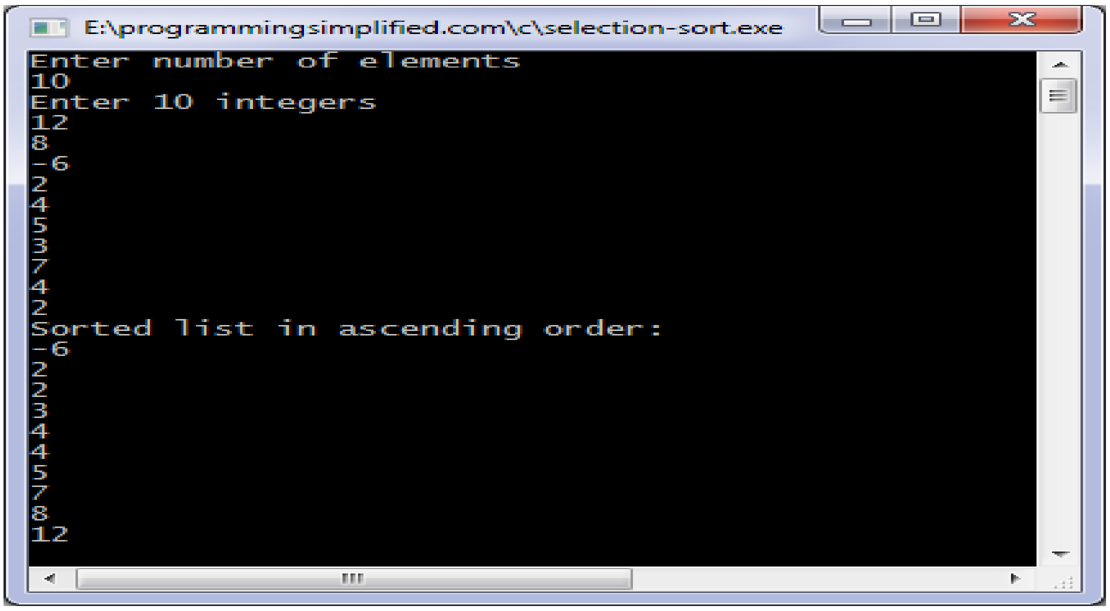
ALGORITHM :

1. Get a list of unsorted numbers
2. Set a marker for the unsorted section at the front of the list
3. Repeat steps 4 - 6 until one number remains in the unsorted section
4. Compare all unsorted numbers in order to select the smallest one
5. Swap this number with the first number in the unsorted section
6. Advance the marker to the right one position

PROGRAM :

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

void main()
{
int array[100], n, c, d, position, 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++ )
{
        position = c;
        for ( d = c + 1 ; d < n ; d++ )
        {
                if ( array[position] > array[d] )
                        position = d;
        }
        if ( position != c )
        {
                swap = array[c];
                array[c] = array[position];
                array[position] = swap;
        }
}
printf("Sorted list in ascending order:\n");
for ( c = 0 ; c < n ; c++ )
        printf("%d\n", array[c]);
getch();
}
```

OUTPUT :



```
Enter number of elements
10
Enter 10 integers
12
8
-6
2
4
5
3
7
4
2
Sorted list in ascending order:
-6
2
2
3
4
4
5
7
8
12
```

EXP. NO : 11(d) **Implementation of searching and sorting algorithms – Bubble Sort**

AIM:

To write a ‘C’ program for the implementation of bubble sort.

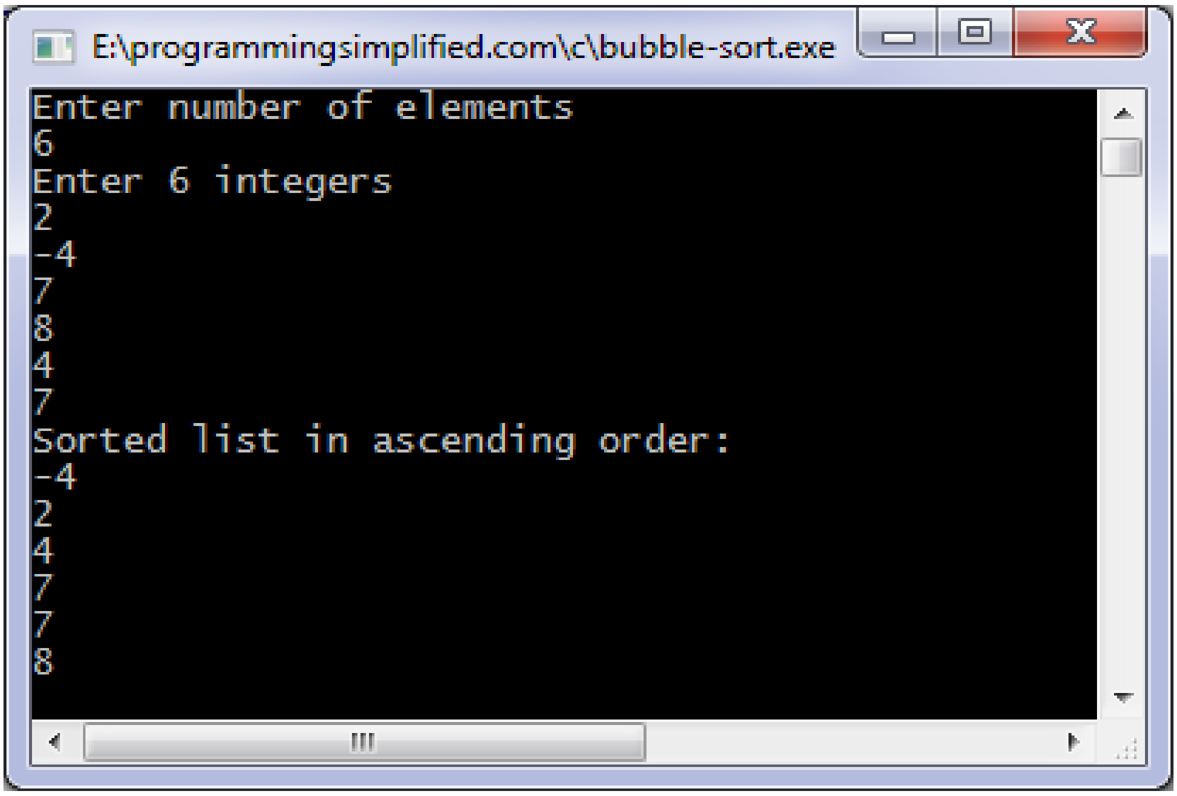
ALGORITHM :

1. Compare adjacent elements. If the first is greater than the second, swap them.
2. Do this for each pair of adjacent elements, starting with the first two and ending with the last two. At this point the last element should be the greatest.
3. Repeat the steps for all elements except the last one.
4. Continue for one less element each time, until there are no more pairs to compare.

PROGRAM :

```
#include <stdio.h>
#include<conio.h>
void 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])
                {
                        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 :



```
Enter number of elements
6
Enter 6 integers
2
-4
7
8
4
7
Sorted list in ascending order:
-4
2
4
7
7
8
```

EXP. NO : 11(e) **Implementation of searching and sorting algorithms – Linear Search**

AIM:

To write a 'C' program for the implementation of linear search.

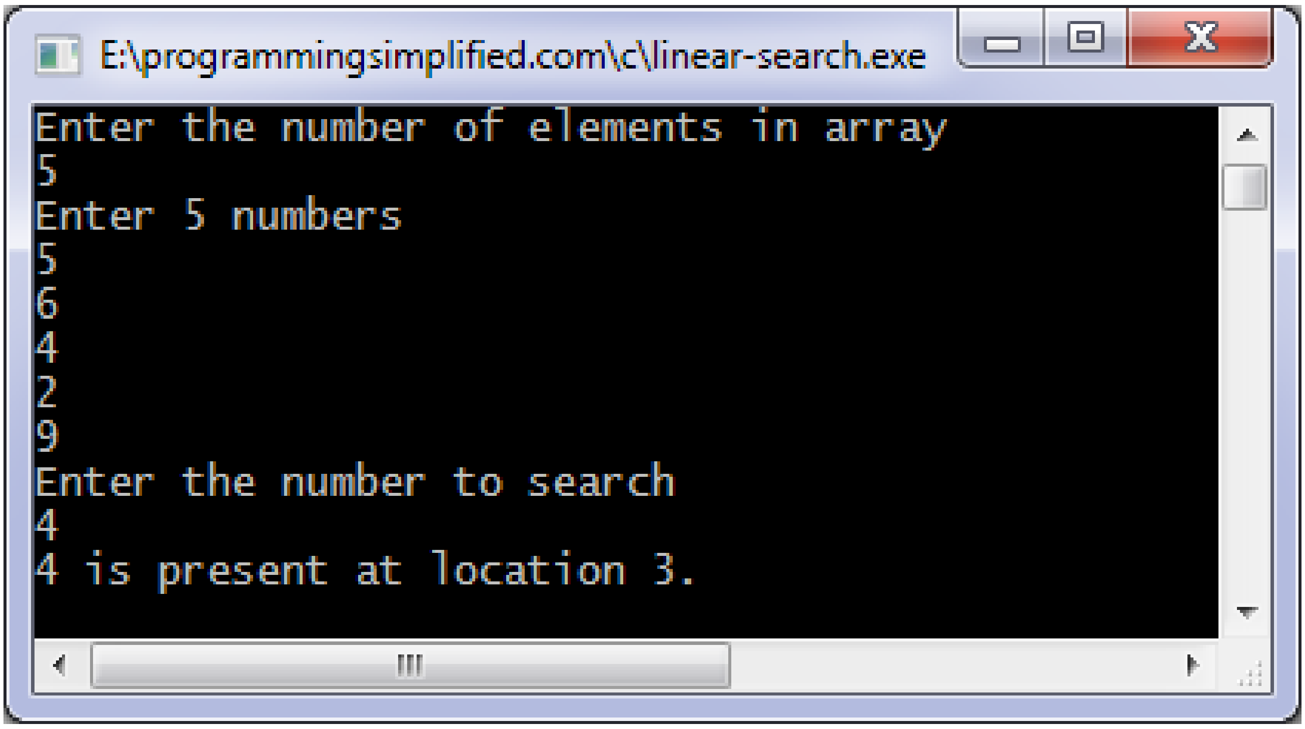
ALGORITHM :

```
1. Repeat this loop:
2. If array[c] = search , then print that the element found and exit from the loop.
3. Else goto Step 1.
4. Print that the element not found in the array.
```

PROGRAM :

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

void  main()
{
int array[100], search, c, n;

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

printf("Enter %d integer(s)\n", n);
for (c = 0; c < n; c++)
        scanf("%d", &array[c]);

printf("Enter the number to search\n");
scanf("%d", &search);

for (c = 0; c < n; c++)
{
        if (array[c] == search)
        {
                printf("%d is present at location %d.\n", search, c+1);
                break;
        }
}
if (c == n)
        printf("%d is not present in array.\n", search);
getch();
}
```

OUTPUT :



```
Enter the number of elements in array
5
Enter 5 numbers
5
6
4
2
9
Enter the number to search
4
4 is present at location 3.
```

EXP. NO : 11(f) **Implementation of searching and sorting algorithms – Binary Search**

AIM:

To write a 'C' program for the implementation of binary search.

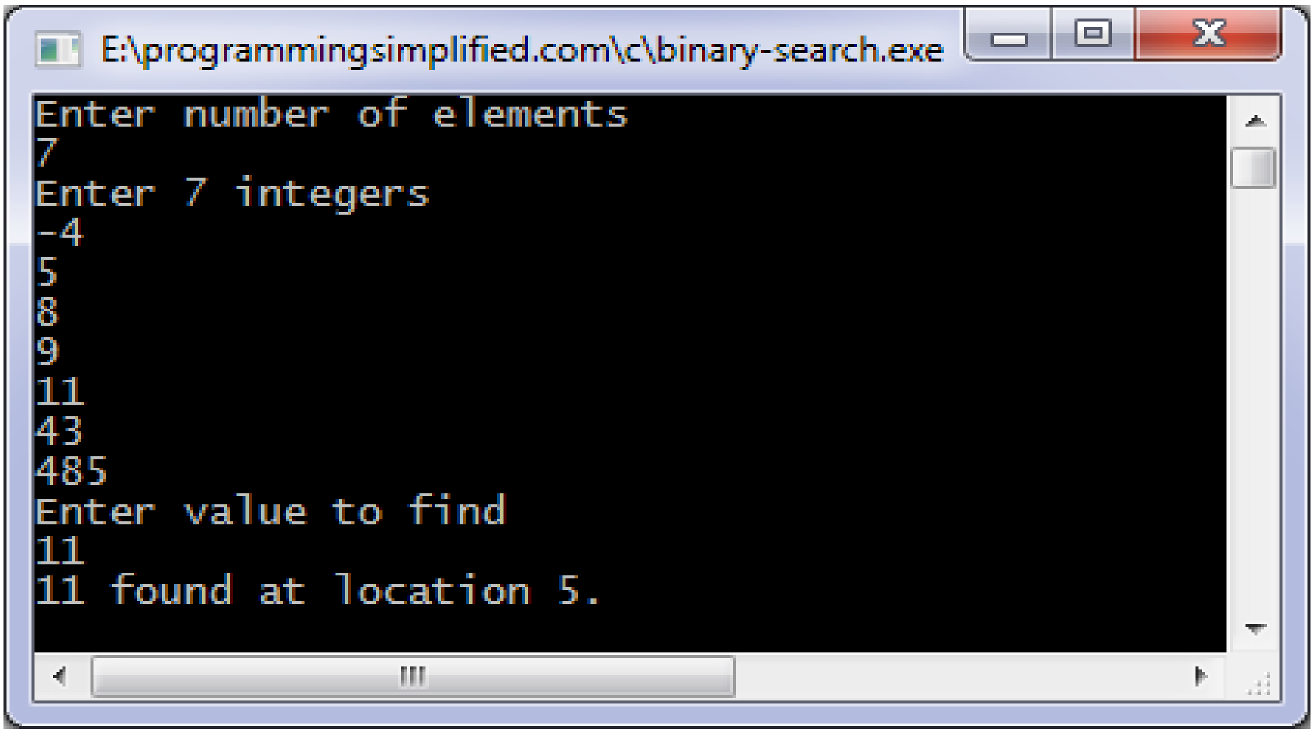
ALGORITHM :

```
1. Declare the required variable.
2. Check conditions first<=last if conditions is true Execute the step 4 until the condition becomes false. If a
   condition is false execute the step 5
3. If key = = mid
        Return mid
4. Else if key > mid
        Return mid-1
5. Else if key < mid
        Return mid+1
6. If ( first > last )
        Print the element is not present in the list
```

PROGRAM :

```
#include <stdio.h>
#include<conio.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 in sorted order\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);
getch();
}
```

OUTPUT :



```
Enter number of elements
7
Enter 7 integers
-4
5
8
9
11
43
485
Enter value to find
11
11 found at location 5.
```

RESULT :

Thus the program to implement the searching and sorting algorithms are executed and the output is verified.

EXP. NO : 12(a) **Implementation of Collision Resolution Technique with Linear Probing**

AIM:

To write a ‘C’ program to implement the Collision Resolution Technique with Linear Probing.

ALGORITHM :

1. Create an array of structure (i.e a hash table).
2. Take a key and a value to be stored in hash table as input.
3. Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.
4. Using the generated index, access the data located in that array index.
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.
6. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item. (Note: This probing will continue until we reach the same element again (from where we began probing)
7. To display all the elements of hash table, element at each index is accessed (via for loop).
8. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements 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<stdlib.h>

/* to store a data (consisting of key and value) in hash table array */
struct item
{
   int key;
   int value;
};

/* each hash table item has a flag (status) and data (consisting of key and value) */
struct hashtable_item
{
   int flag;
   /*     * flag = 0 : data does not exist     * flag = 1 : data exists     * flag = 2 : data existed at least once    */
   struct item *data;
};
struct hashtable_item *array;
int size = 0;
int max = 10;

/* initializing hash table array */
void init_array()
{
   int i;
   for (i = 0; i < max; i++)
   {
            array[i].flag = 0;
            array[i].data = NULL;
   }
}

/* to every key, it will generate a corresponding index */
int hashcode(int key)
{
   return (key % max);
}

/* to insert an element in the hash table */
void insert(int key, int value)
{
   int index = hashcode(key);
   int i = index;

   /* creating new item to insert in the hash table array */
   struct item *new_item = (struct item*) malloc(sizeof(struct item));
   new_item->key = key;
   new_item->value = value;

   /* probing through the array until we reach an empty space */
   while (array[i].flag == 1)
   {
      if (array[i].data->key == key)
      {
            /* case where already existing key matches the given key */
            printf("\n Key already exists, hence updating its value \n");
            array[i].data->value = value;
            return;
      }
      i = (i + 1) % max;
     if (i == index)
     {
     printf("\n Hash table is full, cannot insert any more item \n");
     return;
     }
      }
        array[i].flag = 1;
        array[i].data = new_item;
        size++;
        printf("\n Key (%d) has been inserted \n", key);
     }

/* to remove an element from the hash table */
void remove_element(int key)
{
   int index = hashcode(key);
   int  i = index;

   /* probing through array until we reach an empty space where not even once an element had been present */
   while (array[i].flag != 0)
   {
     if (array[i].flag == 1  &&  array[i].data->key == key )
      {
            // case when data key matches the given key
            array[i].flag =  2;
            array[i].data = NULL;
            size--;
            printf("\n Key (%d) has been removed \n", key);
```

```
                return;
                }
        i = (i + 1) % max;
        if (i == index)
        {
                break;
        }
}
        printf("\n This key does not exist \n");
}

/* to display all the elements of hash table */
void display()
{
    int i;
    for (i = 0; i < max; i++)
    {
                struct item *current = (struct item*) array[i].data;
                if (current == NULL)
                 {
                    printf("\n Array[%d] has no elements \n", i);
                }
                else
               {
                    printf("\n Array[%d] has elements -: \n  %d (key) and %d(value) ", i, current->key, current->value);
                }
          }
       }

int size_of_hashtable()
{
    return size;
}
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 Linear Probing \n\n");
                printf("MENU-: \n1.Inserting item in the Hashtable"
                              "\n2.Removing item from the Hashtable"
                              "\n3.Check the size of Hashtable"
                              "\n4.Display Hashtable"
                                "\n\n Please enter your choice-:");

                scanf("%d", &choice);
                switch(choice)
                {
                        case 1:
                              printf("Inserting element in Hashtable\n");
                              printf("Enter key and value-:\t");
                              scanf("%d %d", &key, &value);
                              insert(key, value);

                              break;
                case 2:

                      printf("Deleting in Hashtable \n Enter the key to delete-:");
                      scanf("%d", &key);
                      remove_element(key);
                      break;

                case 3:

                      n = size_of_hashtable();
                      printf("Size of Hashtable 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 :

```
Implementation of Hash Table in C with Linear Probing

MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 3
Size of Hashtable is-: 0

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


Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 1
Inserting element in Hashtable
Enter key and value-:   12        10

 Key (12) has been inserted

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


Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 1
Inserting element in Hash table
Enter key and value-:   122      4

 Key (122) has been inserted

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


Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 3
Size of Hashtable is-: 2

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

```
Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 4
Array[0] has  no elements

Array[1] has no elements

Array[2] has elements-:
12 (key) and 10 (value)

Array[3] has elements-:
122(key) and 5(value)

Array[4] has no elements

Array[5] has no elements

Array[6] has no elements

Array[7] has no elements

Array[8] has no elements

Array[9] has no elements

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


Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 2
Deleting in Hashtable
Enter the key to delete-: 122

 Key (122) has been removed

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


Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable
```

```
Please enter your choice-: 2
Deleting in Hashtable
Enter the key to delete-: 56

 This key does not exist

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

EXP. NO : 12(b) **Implementation of Collision Resolution Technique with Quadratic Probing**

AIM:

To write a 'C' program to implement the Collision Resolution Technique with Quadratic Probing.

ALGORITHM :

1. Create an array of structure (i.e a hash table).
2. Take a key and a value to be stored in hash table as input.
3. Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.
4. Using the generated index, access the data located in that array index.
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.
6. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item. (Note: This probing will continue until we reach the same element again (from where we began probing) (Note: Here, unlike Linear Probing, probing will be done according to the following formula –
   (currentPosition + h) % arraySize => Linear Probing
   (currentPosition + (h * h)) % arraySize => Quadratic Probing, where h = 1, 2, 3, 4 and so on.)
7. To display all the elements of hash table, element at each index is accessed (via for loop).
8. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements 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<stdlib.h>

/* to store a data (consisting of key and value) in hash table array */
struct item
{
        int key;
        int value;
};

/* each hash table item has a flag (status) and data (consisting of key and value) */
struct hashtable_item
{

        int flag;
        /*   * flag = 0 : data does not exist    * flag = 1 : data exists at given array location
             * flag = 2 : data was present at least once           */
        struct item *data;

};

struct hashtable_item *array;
int size = 0;
int max = 10;

/* this function returns corresponding index of the given key */
int hashcode(int key)
{
        return (key % max);
```

```
}

/* this  function initializes the hash table array */
void init_array()
{
        int i;
        for (i = 0; i < max; i++)
       {
                array[i].flag = 0;
                array[i].data = NULL;
        }
}

/* this function inserts an element in the hash table */
void insert(int key, int value)
{
        int index = hashcode(key);
        int i = index;
        int h = 1;

        struct item *new_item = (struct item*) malloc(sizeof(struct item));
        new_item->key = key;
        new_item->value = value;

        /* probing through the array until an empty space is found */
        while (array[i].flag == 1)
         {
        if (array[i].data->key == key)
         {
                /* case when already present key matches the given key */
                printf("\n This key is already present in hash table, hence updating it's value \n");
                array[i].data->value = value;
                return;
        }
        i = (i + (h * h)) % max;
        h++;
        if (i == index)
         {
                printf("\n Hash table is full, cannot add more elements \n");
                return;
        }
        }
        array[i].flag = 1;
        array[i].data = new_item;
        printf("\n Key (%d) has been inserted\n", key);
        size++;
}

/* to remove an element form the hash table array */
void remove_element(int key)
{
int index = hashcode(key);
int i = index;
int h = 1;
/* probing through the hash table until we reach at location where there had not been an element even once */
while (array[i].flag != 0)
{
        if (array[i].flag == 1  &&  array[i].data->key == key)
        {
                /* case where data exists at the location and its key matches to the given key */
                array[i].flag = 2;
                array[i].data = NULL;
                size--;
                printf("\n Key (%d) has been removed \n", key);
                return;
        }
        i = (i + (h * h)) % max;
        h++;
        if (i == index)
        {
                break;
        }
        }
        printf("\n Key does not exist \n");
}

/* to display the contents of hash table */
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  %d (key) and %d (value) \n", i, array[i].data->key, array[i].data->value);
}
 }

int size_of_hashtable()
{
        return size;
}

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 Quadratic Probing.\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 :

```
Implementation of Hash Table in C with Quadratic Probing
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 Quadratic Probing
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        10

 Key (12) has been inserted

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

Implementation of Hash Table in C with Quadratic Probing
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-:   122       4

 Key (122) has been inserted

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

Implementation of Hash Table in C with Quadratic Probing
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-:   82        5

 Key (82) has been inserted
```

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

Implementation of Hash Table in C with Quadratic Probing
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 Quadratic Probing
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 no elements

Array[2] has elements-:
12 (key) and 10 (value)

Array[3] has elements-:
122(key) and 4(value)

Array[4] has no elements

Array[5] has no elements

Array[6] has no elements

Array[7] has no elements
82(key) and 5(value)
Array[8] has no elements

Array[9] has no elements

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

Implementation of Hash Table in C with Quadratic Probing
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-: 122

 Key (122) has been removed
```

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

Implementation of Hash Table in C with Quadratic Probing
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-: 56

 This key does not exist

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

RESULT :

Thus the program to implement the Graph Collision Resolution Techniques are executed and the output is verified.