ASSIGNMENT NO.1.

Aim :- To create ADT that implement the "set" concept.

a. Add (newElement) -Place a value into the set
b. Remove (element)
c. Contains (element) Return true if element is in collection
d. Size () Return number of values in collection
e. Intersection of two sets
f. Union of two sets
g. Difference between two sets
h.Subset .

Objective:- to study the different set operations.

Theory:-

Sets are a type of abstract data type that allows you to store a list of non-repeated values. Their name derives from the mathematical concept of finite sets.

Unlike an array, sets are unordered and unindexed. You can think about sets as a room full of people you know. They can move around the room, changing order, without altering the set of people in that room. Plus, there are no duplicate people (unless you know someone who has cloned themselves). These are the two properties of a set: the data is unordered and it is not duplicated.

Sets have the most impact in mathematical set theory. These theories are used in many kinds of proofs, structures, and abstract algebra. Creating relations from different sets and codomains are also an important applications of sets.

In computer science, set theory is useful if you need to collect data and do not care about their multiplcity or their order. In databases, especially for relational databases, sets are very useful. There are many commands that finds unions, intersections, and differences of different tables and sets of data.

Program Code:-

```
#include <iostream>

using namespace std;
```

```
const int MAX=50;

template<class T>

class SET

{

        T data[MAX];

        int n;

public:

        SET()

{

                n=-1;

}

        bool insert(T);

        bool remove(T);

        bool contains(T);

        int size();

        void print();

        void input(int num);

        SET unionS(SET,SET);

        SET intersection(SET,SET);

        SET difference(SET,SET);

};


template<class T>
```

```
void SET<T>::input(int num)
{
	T element;
	for(int i=0;i<num;i++)
	{
		cout<<"\nEnter Element: "<<i+1;
		cin>>element;
		insert(element);
	}
}
template<class T>
void SET<T>::print()
{
	for(int i=0;i<=n;i++)
		cout<<" "<<data[i];
}
template<class T>
SET<T> SET<T>::unionS(SET<T> s1,SET<T> s2)
{
	SET<T> s3;

	int flag=0;
	int i=0;
	for(i=0;i<=s1.n;i++)
```

```
	{
		s3.insert(s1.data[i]);
	}
	for(int j=0;j<=s2.n;j++)
	{
		flag=0;
		for(i=0;i<=s1.n;i++)
		{
			if(s1.data[i]==s2.data[j])
			{
				flag=1;
				break;
			}
		}
		if(flag==0)
		{
			s3.insert(s2.data[j]);

		}
	}

	return s3;
}
```

```
template<class T>
SET<T> SET<T>::difference(SET<T> s1,SET<T> s2)
{
	SET<T> s3;
	int flag=1;
	for(int i=0;i<=s1.n;i++)
	{
		for(int j=0;j<=s2.n;j++)
		{
			if(s1.data[i]==s2.data[j])
			{
				flag=0;
				break;
			}
			else flag=1;
		}
		if(flag==1)
		{
			s3.insert(s1.data[i]);

		}
	}
	return s3;
}
```

```
template<class T>
SET<T> SET<T>::intersection(SET<T> s1,SET<T> s2)
{
	SET<T> s3;
	for(int i=0;i<=s1.n;i++)
	{
		for(int j=0;j<=s2.n;j++)
		{
			if(s1.data[i]==s2.data[j])
			{
				s3.insert(s1.data[i]);
				break;
			}
		}
	}
	return s3;
}
template<class T>
bool SET<T>::insert(T element)
{
	if(n>=MAX)
	{
		cout<<"\nOverflow.SET is full.\n";
```

```
        return false;
    }
    data[++n]=element;
    return true;
}

template<class T>
bool SET<T>::remove(T element)
{
    if(n==-1)
    {
        cout<<"Underflow. Cannot perform delete operation on empty SET.";
        return false;
    }
    for(int i=0;i<=n;i++)
    {
        if(data[i]==element)
        {
            for(int j=i;i<=n;j++)
            {
                data[j]=data[j+1];
            }
            return true;
        }
```

```
	}
	//data[n--]=0;
	return false;
}
template<class T>
bool SET<T>::contains(T element)
{
	for(int i=0;i<=n;i++)
	{
		if(data[i]==element)
			return true;
	}
	return false;
}
template<class T>
int SET<T>::size()
{
	return n+1;
}
int main() {

	SET<int> s1,s2,s3;
	int choice;
	int element;
```

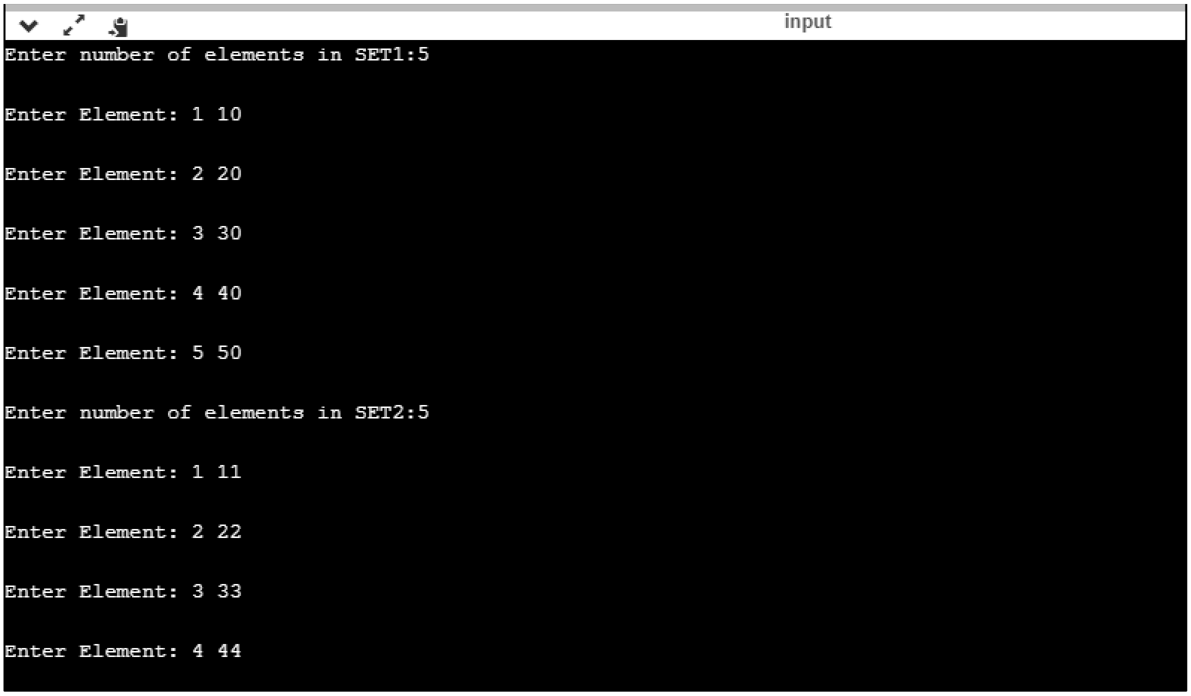
```
cout<<"\nEnter number of elements in SET1:";
cin>>element;//element is used for taking size
s1.input(element);
cout<<"\nEnter number of elements in SET2:";
cin>>element;//element is used for taking size
s2.input(element);
do
{
    cout<<"\n***** SET OPERATIONS *****"
            <<"\n1.Insert"
            <<"\n2.Remove"
            <<"\n3.Search"
            <<"\n4.Size of Set"
            <<"\n5.Intersection"
            <<"\n6.Union"
            <<"\n7.Difference"
            <<"\n8.Check if Subset"
            <<"\nEnter Your Choice: ";
    cin>>choice;
    switch(choice)
    {
    case 1:
        cout<<"\nEnter Element: ";
        cin>>element;
```

```
        if(s1.insert(element))
        {
            cout<<element<<" inserted";
        }
        else
        {
            cout<<"Insertion Failed";
        }
        break;
    case 2:
        cout<<"\nEnter Element: ";
        cin>>element;
        if(s1.remove(element))
        {
            cout<<element<<" deleted";
        }
        else
        {
            cout<<"Deletion Failed";
        }
        break;
    case 3:
        cout<<"\nEnter Element: ";
        cin>>element;
```

```
        if(s1.contains(element))
        {
                cout<<element<<" is present";
        }
        else
        {
                cout<<element<<"is not  Present";
        }
        break;
case 4:
        cout<<"\nSize = "<<s1.size();
        break;
case 5:
        s3=s1.intersection(s1,s2);
        cout<<"\nSET 1's elements: ";
        s1.print();
        cout<<"\nSET 2's elements: ";
        s2.print();
        cout<<"\nIntersection: :";
        s3.print();
        break;

case 6:
```

```
                s3=s1.unionS(s1,s2);
                cout<<"\nSET 1's elements: ";
                s1.print();
                cout<<"\nSET 2's elements: ";
                s2.print();
                cout<<"\nUnion :";
                s3.print();
                break;
            case 7:
                s3=s1.difference(s1,s2);
                cout<<"\nSET 1's elements: ";
                s1.print();
                cout<<"\nSET 2's elements: ";
                s2.print();
                cout<<"\nDifference :";
                s3.print();
                break;

            }
        }while(choice!=0);
        return 0;
}
```

Output Screenshots:-



```
input
Enter number of elements in SET1:5

Enter Element: 1 10

Enter Element: 2 20

Enter Element: 3 30

Enter Element: 4 40

Enter Element: 5 50

Enter number of elements in SET2:5

Enter Element: 1 11

Enter Element: 2 22

Enter Element: 3 33

Enter Element: 4 44
```

```
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
6.Union
7.Difference
8.Check if Subset
Enter Your Choice: 1

Enter Element: 22
22 inserted
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
6.Union
7.Difference
8.Check if Subset
```

```
8.Check if Subset
Enter Your Choice: 5

SET 1's elements:  10 20 30 40 50 22
SET 2's elements:  11 22 33 44 55
Intersection: : 22
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
6.Union
7.Difference
8.Check if Subset
Enter Your Choice: 6

SET 1's elements:  10 20 30 40 50 22
SET 2's elements:  11 22 33 44 55
Union : 10 20 30 40 50 22 11 33 44 55
***** SET OPERATIONS *****
```

Conclusion:- Thus,we have studied different operations on set ADT.

ASSIGNMENT NO.2.

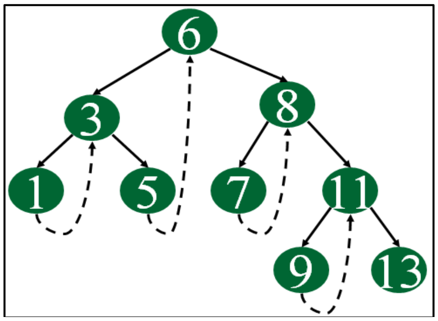
Aim :- Construct a threaded binary search tree by inserting values in the given order and traverse it in inorder traversal using threads.

Objective:- To study the concept of threaded binary tree**.**

Theory:-

Inorder traversal of a Binary tree can either be done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).
There are two types of threaded binary trees.
Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)
Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
The threads are also useful for fast accessing ancestors of a node.

Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



Algorithm:-

Let **tmp be the newly inserted node**. There can be three cases during insertion:

Case 1: Insertion in empty tree

Both left and right pointers of tmp will be set to NULL and new node becomes the root.

```
root = tmp;

tmp -> left = NULL;

tmp -> right = NULL;
```

Case 2: When new node inserted as the left child

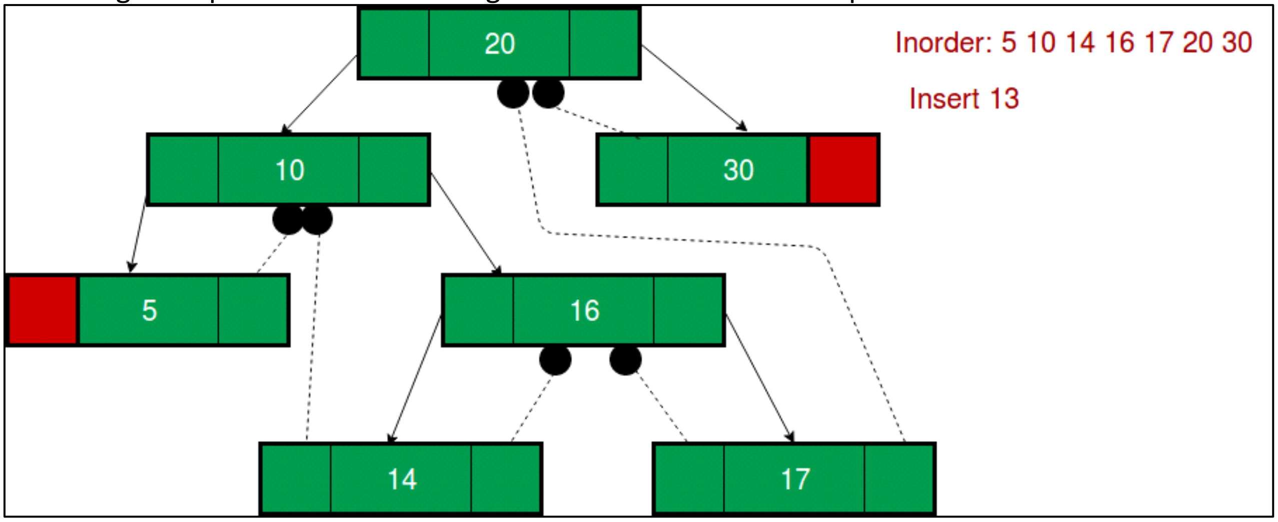
After inserting the node at its proper place we have to make its left and right threads points to inorder predecessor and successor respectively. The node which was inorder successor. So the left and right threads of the new node will be-

```
tmp -> left = par ->left;

tmp -> right = par;
```

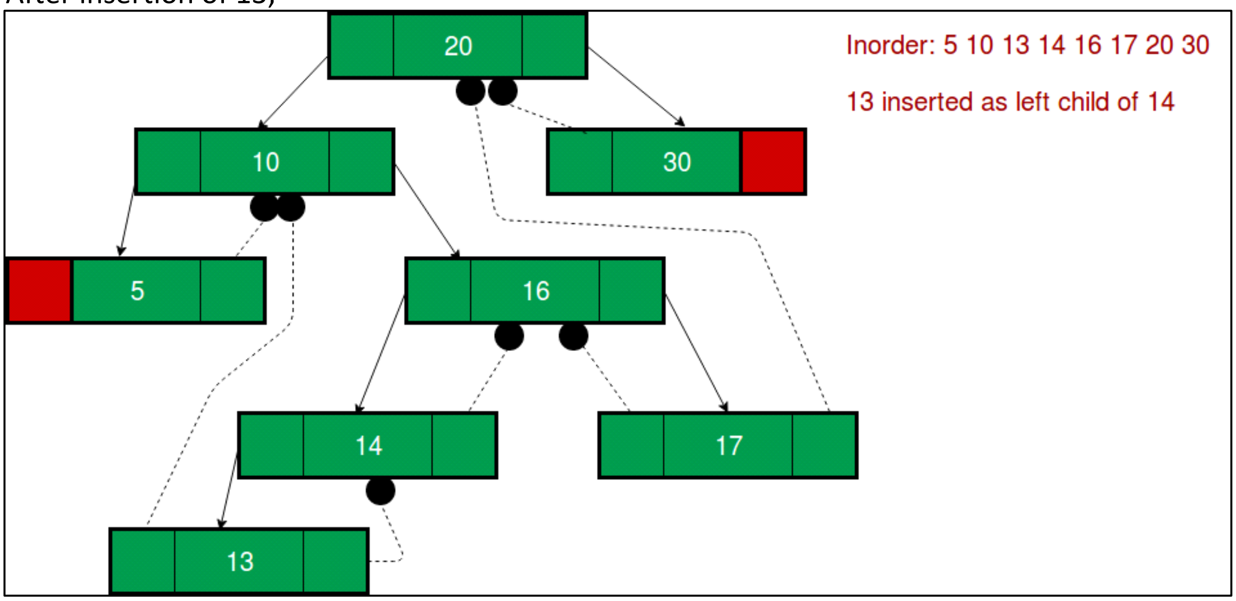
Before insertion, the left pointer of parent was a thread, but after insertion it will be a link pointing to the new node.

```
par -> lthread = false;

par -> left = temp;
```

Following example show a node being inserted as left child of its parent.



After insertion of 13,



Predecessor of 14 becomes the predecessor of 13, so left thread of 13 points to 10.
Successor of 13 is 14, so right thread of 13 points to left child which is 13.
Left pointer of 14 is not a thread now, it points to left child which is 13.

Case 3: When new node is inserted as the right child

The parent of tmp is its inorder predecessor. The node which was inorder successor of the parent is now the inorder successor of this node tmp. So the left and right threads of the new node will be-

tmp -> left = par;

tmp -> right = par -> right;

Before insertion, the right pointer of parent was a thread, but after insertion it will be a link pointing to the new node.

par -> rthread = false;

par -> right = tmp;

Program Code:-

```
#include <iostream>
using namespace std;
class TBT;
class node
{
	node *left,*right;
	int data;
	bool rbit,lbit;
public:
	node()
{
		left=NULL;
		right=NULL;
		rbit=lbit=0;
}
	node(int d)
	{
		left=NULL;
		right=NULL;
```

```
        rbit=lbit=0;
        data=d;
    }
    friend class TBT;
};

class TBT
{
    node *root; //acts as a dummy node
public:
    TBT() //dummy node initialization
{
        root=new node(9999);
        root->left=root;
        root->rbit=1;
        root->lbit=0;
        root->right=root;
}
    void create();
    void insert(int data);
    node *inorder_suc(node *);
    void inorder_traversal();
    node * preorder_suc(node *c);
    void preorder_traversal();
```

```
};
//--------------------------------------------
void TBT::preorder_traversal()
{
	node *c=root->left;
	while(c!=root)
	{
		cout<<" "<<c->data;
		c=preorder_suc(c);
	}
}
void TBT::inorder_traversal()
{
	node *c=root->left;
	while(c->lbit==1)
		c=c->left;
	while(c!=root)
	{
		cout<<" "<<c->data;
		c=inorder_suc(c);
	}
}
node* TBT::inorder_suc(node *c)
{
```

```
        if(c->rbit==0)
                return c->right;
        else
                c=c->right;
        while(c->lbit==1)
        {
                c=c->left;
        }
        return c;
}
node *TBT::preorder_suc(node *c)
{
        if(c->lbit==1)
        {
                return c->left;
        }
        while(c->rbit==0)
        {
                c=c->right;
        }
        return c->right;
}
//-------- Create Method
void TBT::create()
```

```
{

        int n;
        if(root->left==root&&root->right==root)
        {
        cout<<"\nEnter number of nodes:";
        cin>>n;
        for(int i=0;i<n;i++)
        {
                int info;
                cout<<"\nEnter data: ";
                cin>>info;
                this->insert(info);
        }
        }
        else
        {
                cout<<"\nTree is Already created.\n";
        }
}
void TBT::insert(int data)
{

        if(root->left==root&&root->right==root) //no node in tree
```

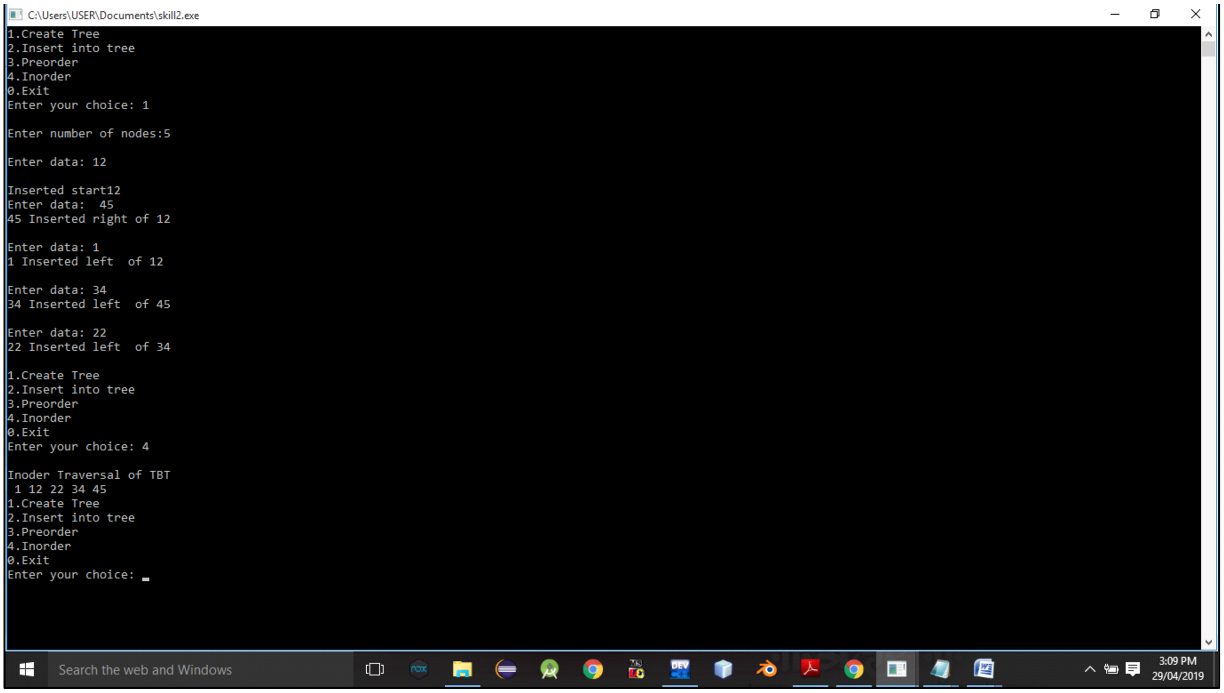
```
{
        node *p=new node(data);
        p->left=root->left;
        p->lbit=root->lbit; //0
        p->rbit=0;
        p->right=root->right;
        root->left=p;
        root->lbit=1;
        cout<<"\nInserted start"<<data;
        return;
}
node *cur=new node;
cur=root->left;
while(1)
{

        if(cur->data<data)   //insert right
        {
                node *p=new node(data);
                if(cur->rbit==0)
                {
                        p->right=cur->right;
                        p->rbit=cur->rbit;
                        p->lbit=0;
```

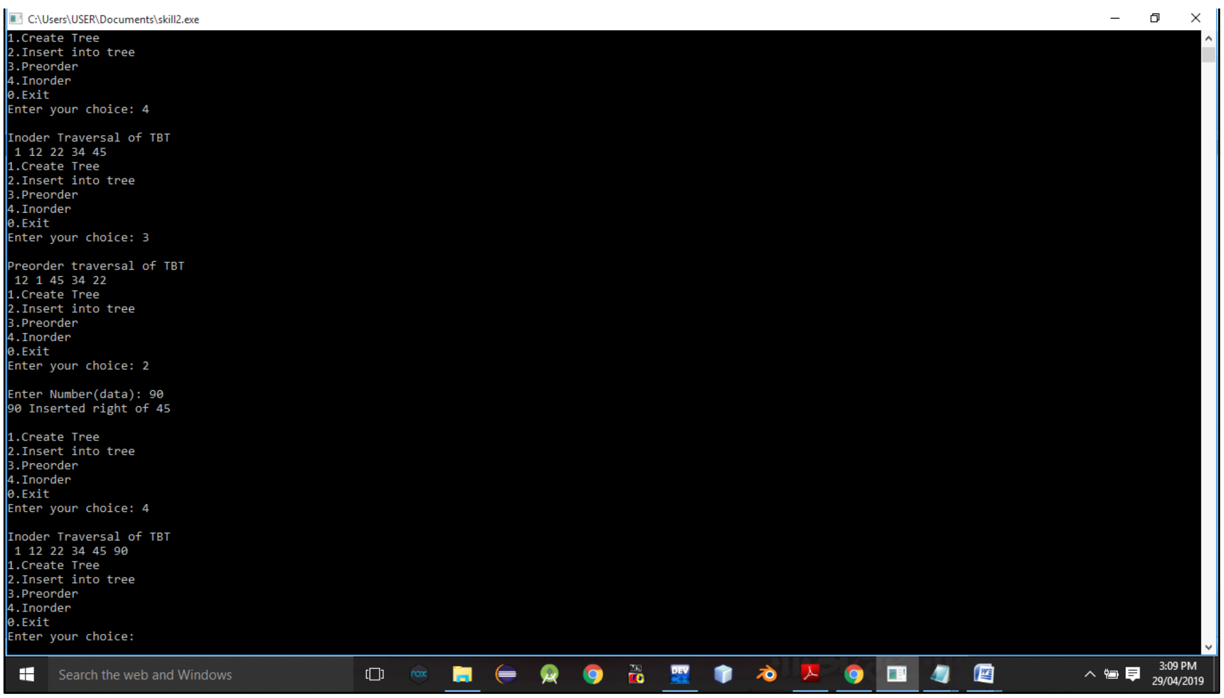
```
                p->left=cur;
                cur->rbit=1;
                cur->right=p;
                //cout<<"\nInserted right "<<data;
                cout<< data<<" Inserted right"<<" of "<< cur->data<<endl;
                return;
            }
            else
                cur=cur->right;
        }
        if(cur->data>data) //insert left
        {
            node *p=new node(data);
            if(cur->lbit==0)
            {
                p->left=cur->left;
                p->lbit=cur->lbit;
                p->rbit=0;
                p->right=cur; //successor
                cur->lbit=1;
                cur->left=p;
                cout<<data <<" Inserted left "<<" of "<<cur->data<<endl;
                return;
            }
```

```
                        else
                                cur=cur->left;
                }
        }
}

int main() {
        TBT t1;
        int value;
        int choice;
        do
        {
                cout<<"\n1.Create Tree\n2.Insert into
tree\n3.Preorder\n4.Inorder\n0.Exit\nEnter your choice: ";
                cin>>choice;
                switch(choice)
                {
                case 1:
                        t1.create();
                        break;
                case 2:
```

```
                cout<<"\nEnter Number(data): ";
                cin>>value;
                t1.insert(value);
                break;
            case 3:
                cout<<"\nPreorder traversal of TBT\n";
                t1.preorder_traversal();
                break;
            case 4:
                cout<<"\nInoder Traversal of TBT\n";
                t1.inorder_traversal();
                break;
            default:
                cout<<"\nWrong choice";
            }

    }while(choice!=0);



    return 0;
}
```

Output Screenshots:-





Conclusion:- Thus,we have studied Threaded binary tree.

ASSIGNMENT NO.3.

Aim :-

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representations used .

Objective:- To learn adjacency list representation of graph.

Theory:-

Following two are the most commonly used representations of a graph.
1. Adjacency Matrix
2. Adjacency List
There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Adjacency List:
An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is adjacency list representation of the above graph.

Algorithm:-

1.BFS :-

- **Step 1:** SET STATUS = 1 (ready state)
  for each node in G
- **Step 2:** Enqueue the starting node A
  and set its STATUS = 2
  (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until
  QUEUE is empty

- **Step 4:** Dequeue a node N. Process it
  and set its STATUS = 3
  (processed state).
- **Step 5:** Enqueue all the neighbours of
  N that are in the ready state
  (whose STATUS = 1) and set
  their STATUS = 2
  (waiting state)
  [END OF LOOP]
- **Step 6:** EXIT

2.DFS :-

- **Step 1:** SET STATUS = 1 (ready state) for each node in G
- **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until STACK is empty
- **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
- **Step 5:** Push on the stack all the neighbours of N that are in the ready state
  (whose STATUS = 1) and set their
  STATUS = 2 (waiting state)
  [END OF LOOP]
- **Step 6:** EXIT

Program Code:-

```
#include <iostream>

#include<iomanip>

using namespace std;


const int MAX=30;
```

```
class Queue //Queue for BFS TRAVERSAL
{
        int front,rear;
        string data[MAX];
public:
        Queue()
{
                front=-1;
                rear=-1;
}
        bool empty()
        {
                if(rear==-1)
                        return 1;
                else
                        return 0;
        }
        bool inqueue(string str)
        {
                if(front==-1 && rear==-1)
                {
                        front=rear=0;
                        data[rear]=str;
                        return true;
                }
                else
```

```
        {
            rear=rear+1;
            data[rear]=str;
            return true;
        }
    }
    string dequeue()
    {
        string  p;
        if(front==rear)
        {
            p=data[front];
            front=-1;
            rear=-1;
        }
        else
        {
            p=data[front];
            front=front+1;
        }
        return p;
    }

};

class node //node class for each airport
{
```

```
        node *next;
        string city;
        int timeCost;
public:
        friend class graph;
        node()
        {
                next=NULL;
                city="";
                timeCost=-1;
        }
        node(string city,int weight)
        {
                next=NULL;
                this->city=city;
                timeCost=weight;
        }
};
class graph //Contains total graph of airports
{
        node *head[MAX];
        int n;
        int visited[MAX];
public:
        graph(int num)
{
                n=num;
```

```
        for(int i=0;i<n;i++)
            head[i]=NULL;
}
    void insert(string city1,string city2,int time);
    void insertUndirected(string city1,string city2,int time);
    void readdata(int gType);
    int getindex(string s1);
    void outFlights();
    void inFlights();
    void DFS(string str);
    void BFS();
    void dfsTraversal();

};
void graph::BFS()
{
    string str=head[0]->city;
    int j;
    //node *p;
    for(int i=0;i<n;i++)
        visited[i]=0;
    Queue queue;
    queue.inqueue(str);
    int i=getindex(str);
        cout<<"BFS Traversal: \n";
        cout<<" "<<str<<" ";
        visited[i]=1;
```

```
        while(!queue.empty())
        {
            string p=queue.dequeue();
            i=getindex(p);



            //visited[i]=1;

            for(node *q=head[i];q!=NULL;q=q->next)
            {
                i=getindex(q->city);
                str=q->city;
                if(!visited[i])
                {
                    queue.inqueue(q->city);
                    visited[i]=1;
                    cout<<" "<<str<<" ";
                }
            }
        }
        cout<<"\n";
}
void graph::dfsTraversal()
    {
        for( int i=0;i<n;i++)
            visited[i]=0;
        cout<<"\nDFS TRAVERSAL: \n";
```

```
        DFS(head[0]->city);
        cout<<"\n";
    }
void graph::DFS(string str)
{
    node *p;
    int i=getindex(str);

    cout<<" "<<str<<" ";
    p=head[i];
    visited[i]=1;
    while(p!=NULL)
    {
        str=p->city;
        i=getindex(str);
        if(!visited[i])
            DFS(str);
        p=p->next;
    }

}
void graph::inFlights()
{
    int count[n];
    for(int i=0;i<n;i++)
        count[i]=0;
    cout<<"====== In degree =========\n";
```

```
	for(int i=0;i<n;i++)
	{
		cout<<"\n"<<setw(8)<<"Source"<<setw(8)<<"Destin."<<setw(8)<<"Time";
		for(int j=0;j<n;j++)
		{
			node *p=head[j]->next;
			while(p!=NULL)
			{
				if(p->city==head[i]->city)
				{
					count[i]=count[i]+1;
					cout<<"\n"<<setw(8)<<head[j]-
>city<<setw(8)<<head[i]->city<<setw(8)<<p->timeCost;
				}

				p=p->next;
			}
		}
		cout<<"\nFlights to "<<head[i]->city<<" = "<<count[i]<<endl;
		cout<<"------------------------------------\n";
	}

}
void graph::outFlights()
{
	int count;
	for(int i=0;i<n;i++)
	{
```

```
		node *p=head[i]->next;
		count=0;
		cout<<"\n"<<setw(8)<<"Source"<<setw(8)<<"Destin."<<setw(8)<<"Time";
		if(p==NULL)
		{
			cout<<"\nNo Flights from "<<head[i]->city;
		}
		else
		{
			while(p!=NULL)

			{
				cout<<"\n"<<setw(8)<<head[i]->city<<setw(8)<<p-
>city<<setw(8)<<p->timeCost;
				count++;
				p=p->next;
			}
		}
		cout<<"\nNo. of flights: "<<count<<endl;;
		cout<<"-------------------------------------\n";
	}
}
int graph::getindex(string s1)
{
	for(int i=0;i<n;i++)
	{
		if(head[i]->city==s1)
			return i;
```

```
	}
	return -1;
}
void graph::insert(string city1,string city2,int time)
{
	node *source;
	node *dest=new node(city2,time);

	int ind=getindex(city1); //for getting head nodes index in array
	if(head[ind]==NULL)
		head[ind]=dest;
	else
	{
		source=head[ind];
		while(source->next!=NULL)
			source=source->next;
		source->next=dest;
	}
}
void graph::insertUndirected(string city1,string city2,int time)
{
	node *source;
	node *dest=new node(city2,time);
	node *dest2=new node(city1,time); //for second flight insertion

	int ind=getindex(city1); //for getting head nodes index in array
	int ind2=getindex(city2);
```

```
/*	if(head[ind]==NULL && head[ind2]==NULL) //when no flights in graph
	{
		head[ind]=dest;
		head[ind2]=dest2;
	}
	else if(head[ind]==NULL && head[ind2]!=NULL) //no flight in first list but flight in
second list
	{
		head[ind]=dest; //inserted first flight
		source=head[ind2];
		while(source->next!=NULL)
			source=source->next;
		source->next=dest2;
	}
	else if(head[ind]!=NULL && head[ind2]==NULL)
	{
		head[ind2]=dest2; //inserted first flight
		source=head[ind];
		while(source->next!=NULL)
			source=source->next;
		source->next=dest;
	}
	else
	{*/
		source=head[ind];
		while(source->next!=NULL)
			source=source->next;
		source->next=dest;
```

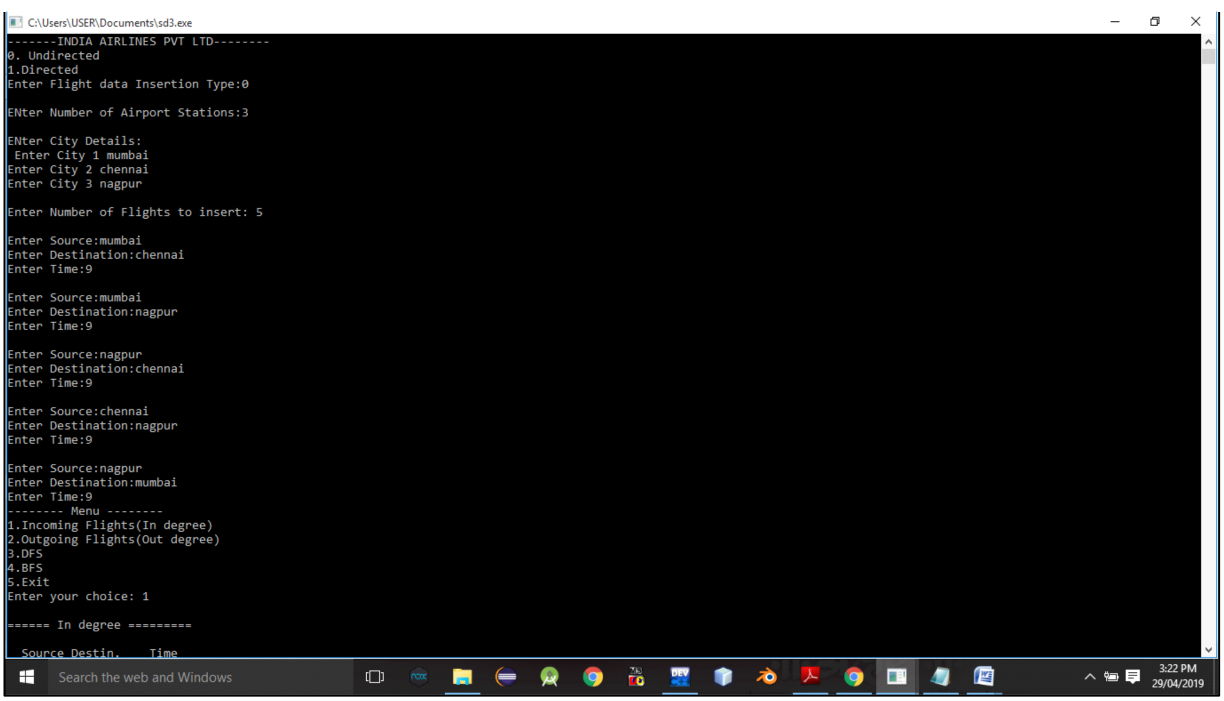
```
            source=head[ind2];
            while(source->next!=NULL)
                source=source->next;
            source->next=dest2;

        //}
}
void graph::readdata(int gType)
{
        string city1,city2,tmpcity;
        int fcost;
        int flight;
        cout<<"\nENter City Details:\n ";
        for(int i=0;i<n;i++)
        {
            head[i]=new node;
            cout<<"Enter City "<<i+1<<" ";
            cin>>tmpcity;
            head[i]->city=tmpcity;
        }
        cout<<"\nEnter Number of Flights to insert: ";
        cin>>flight;
        if(gType==1)
        {
            for(int i=0;i<flight;i++)
            {
```

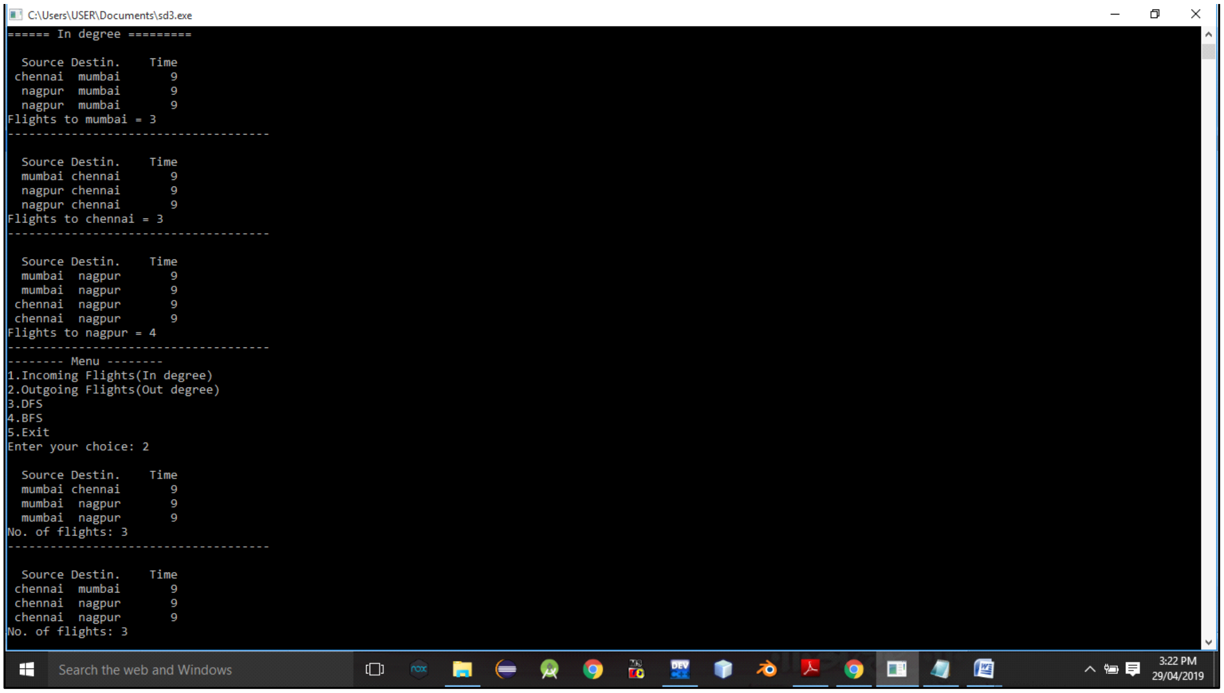
```
                cout<<"\nEnter Source:";
                cin>>city1;
                cout<<"Enter Destination:";
                cin>>city2;
                cout<<"Enter Time:";
                cin>>fcost;
                insert(city1,city2,fcost);
            }
        }
        else
        {
            for(int i=0;i<flight;i++)
            {
                cout<<"\nEnter Source:";
                cin>>city1;
                cout<<"Enter Destination:";
                cin>>city2;
                cout<<"Enter Time:";
                cin>>fcost;
                insertUndirected(city1,city2,fcost);
                //cout<<"\ninserted"<<i+1;
            }
        }
}
int main() {
        int number,choice;
        int graphype;
```

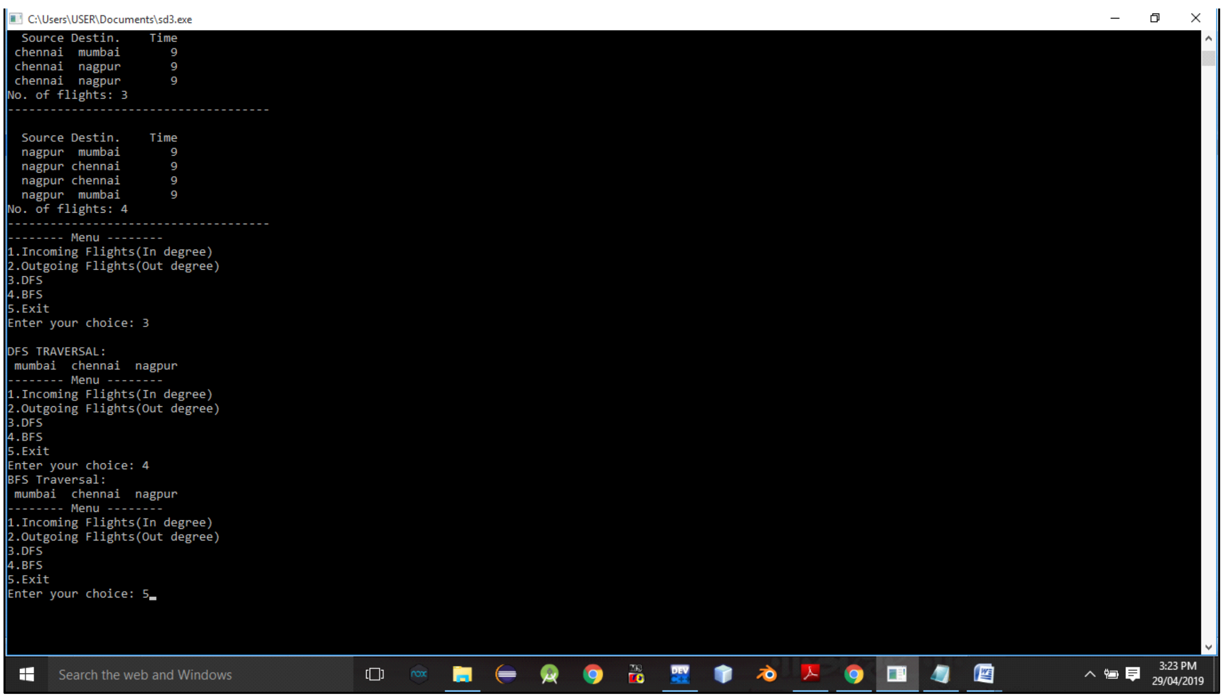
```
cout<<"-------INDIA AIRLINES PVT LTD--------"
        <<"\n0. Undirected\n1.Directed\nEnter Flight data Insertion Type:";
cin>>graphype;
cout<<"\nENter Number of Airport Stations:";
cin>>number;
graph g1(number);
g1.readdata(graphype);
do
{
        cout<<"-------- Menu --------"
                        <<"\n1.Incoming Flights(In degree)"
                        <<"\n2.Outgoing Flights(Out degree)"
                        <<"\n3.DFS"
                        <<"\n4.BFS"
                        <<"\n5.Exit"
                        <<"\nEnter your choice: ";
        cin>>choice;
        switch(choice)
        {
        case 1:
                cout <<"" << endl;
                g1.inFlights();
                break;
        case 2:
                g1.outFlights();
                break;
        case 3:
```

```
                    g1.dfsTraversal();
                    break;
                case 4:
                    g1.BFS();
                    break;
                default:
                    cout<<"\nWrong Choice";
                }
        }while(choice!=5);



        return 0;
}
```

Output Screenshots:-







Conclusion:- Thus,we have studied adjacency graph representation.

ASSIGNMENT NO.4.

Aim :- For a weighted graph G, find the minimum spanning tree using Prims algorithm .

Objective:- To study the prims algorithm.

Theory:-

Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.
A group of edges that connects two set of vertices in a graph is called cut in graph theory. So, at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the verices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices)
.
How does Prim's Algorithm Work? The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

Algorithm:-

1) Create a set mstSet that keeps track of vertices already included in MST.

2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3) While mstSet doesn't include all vertices
….**a)** Pick a vertex u which is not there in mstSet and has minimum key value.
….**b)** Include u to mstSet.
….**c)** Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

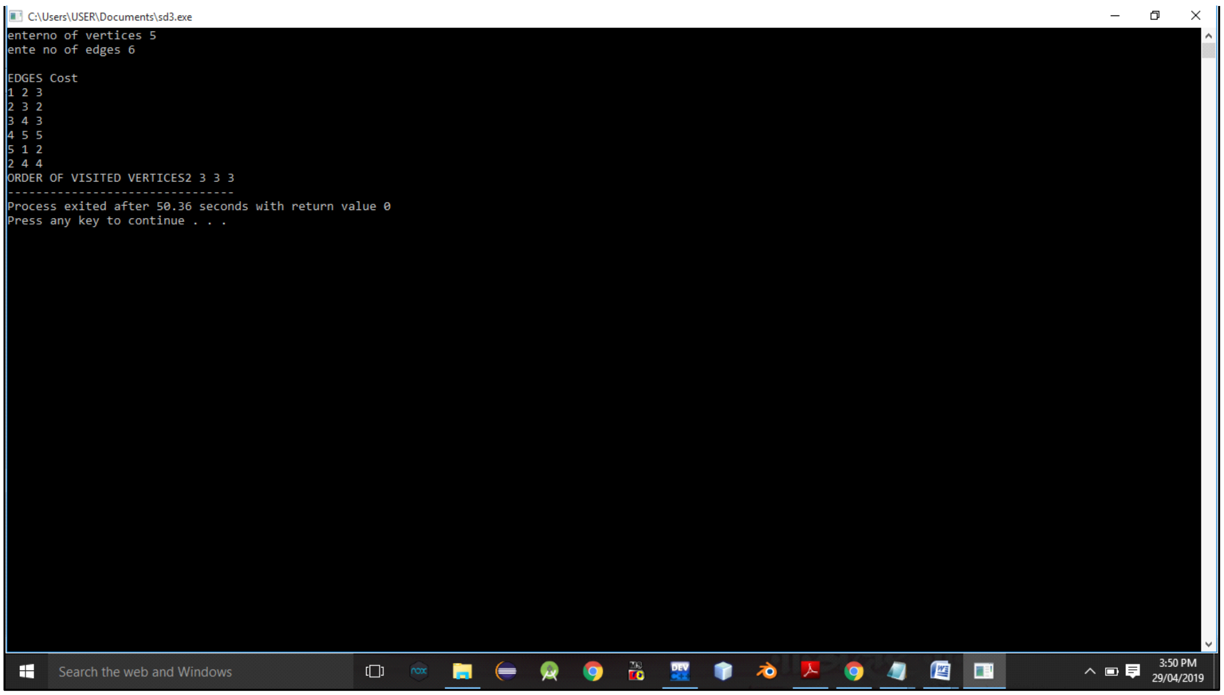
Program Code:-

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,stk[10],top,v,visit[10],visited[10],u;

main()
{
	int m,c;
	cout <<"enterno of vertices";
	cin >> n;
	cout <<"ente no of edges";
	cin >> m;
	cout <<"\nEDGES Cost\n";
	for(k=1;k<=m;k++)
	{
		cin >>i>>j>>c;
		cost[i][j]=c;
	}
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
		if(cost[i][j]==0)
		cost[i][j]=31999;
```

```
cout <<"ORDER OF VISITED VERTICES";
k=1;
while(k<n)
{
        m=31999;
        if(k==1)
        {
                for(i=1;i<=n;i++)
                        for(j=1;j<=m;j++)
                        if(cost[i][j]<m)
                        {
                                m=cost[i][j];
                                u=i;
                        }
        }
        else
        {
                for(j=n;j>=1;j--)
                if(cost[v][j]<m && visited[j]!=1 && visit[j]!=1)
                {
                        visit[j]=1;
                        stk[top]=j;
                        top++;
                        m=cost[v][j];
                        u=j;
                }
```

```
            }
            cost[v][u]=31999;
            v=u;
            cout<<v << " ";
            k++;
            visit[v]=0; visited[v]=1;
        }
}
```

Output Screenshots:-



Conclusion:- Thus,we have studied prims algorithm.

ASSIGNMENT NO.5.

Aim :-

You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures .

Objective:- To study the use of kruskal's and prims algorithm in given problem.

Theory:- *What is Minimum Spanning Tree?*

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

How many edges does a minimum spanning tree has?

A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

What are the applications of Minimum Spanning Tree?

See this for applications of MST.

Below are the steps for finding MST using Kruskal's algorithm

1. *Sort all the edges in non-decreasing order of their weight.*
2. *Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.*
3. *Repeat step#2 until there are (V-1) edges in the spanning tree.*

The step#2 uses Union-Find algorithm to detect cycle.

The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Algorithm:-

- create a forest *F* (a set of trees), where each vertex in the graph is a separate tree
- create a set *S* containing all the edges in the graph
- while *S* is nonempty and *F* is not yet spanning
  - remove an edge with minimum weight from *S*
  - if the removed edge connects two different trees then add it to the forest *F*, combining two trees into a single tree

At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

Program Code:-

```
#include <iostream>

#include<iomanip>

using namespace std;


const int MAX=10;

class EdgeList;  //forward declaration

class Edge              //USED IN KRUSKAL

{

        int u,v,w;

public:

        Edge(){} //Empty Constructor

        Edge(int a,int b,int weight)

        {

                u=a;

                v=b;

                w=weight;

        }

        friend class EdgeList;

        friend class PhoneGraph;

};

//---- EdgeList Class ----------
```

```
class EdgeList
{
	Edge data[MAX];
	int n;
public:
	friend class PhoneGraph;
	EdgeList()
	{ n=0;}
	void sort();
	void print();

};
//----Bubble Sort for sorting edges in increasing weights' order ---//
void EdgeList::sort()
{
	Edge temp;
	for(int i=1;i<n;i++)
		for(int j=0;j<n-1;j++)
			if(data[j].w>data[j+1].w)
			{
				temp=data[j];
				data[j]=data[j+1];
				data[j+1]=temp;
			}
}
```

```
void EdgeList::print()
{
	int cost=0;
	for(int i=0;i<n;i++)
	{
		cout<<"\n"<<i+1<<" "<<data[i].u<<"--"<<data[i].v<<" = "<<data[i].w;
		cost=cost+data[i].w;
	}
	cout<<"\nMinimum cost of Telephone Graph = "<<cost;
}
//------------ Phone Graph Class---------------
class PhoneGraph
{
	int data[MAX][MAX]={{0, 28, 0, 0, 0,10,0},
	 {28,0,16,0,0,0,14},
	 {0,16,0,12,0,0,0},
	 {0,0,12,0,22,0,18},
	 {0,0,0,22,0,25,24},
	 {10,0,0,0,25,0,0},
	 {0,14,0,18,24,0,0},
	};
	int n;

public:
	PhoneGraph(int num)
```

```
{
        n=num;
}
    void readgraph();
    void printGraph();
    int mincost(int cost[],bool visited[]);
    int prim();
    void kruskal(EdgeList &spanlist);
    int find(int belongs[], int vertexno);
    void unionComp(int belongs[], int c1,int c2);
};
void PhoneGraph::readgraph()
{
    cout<<"Enter Adjacency(Cost) Matrix: \n";
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n; j++)
            cin>>data[i][j];
    }
}
void PhoneGraph::printGraph()
{
    cout<<"\nAdjacency (COST) Matrix: \n";
    for(int i=0;i<n;i++)
    {
```

```
                for(int j=0;j<n;j++)
                {
                        cout<<setw(3)<<data[i][j];
                }
                cout<<endl;
        }
}
int PhoneGraph::mincost(int cost[],bool visited[]) //finding vertex with minimum cost
{
        int min=9999,min_index; //initialize min to MAX value(ANY) as temporary
        for(int i=0;i<n;i++)
        {
                if(visited[i]==0 && cost[i]<min)
                {
                        min=cost[i];
                        min_index=i;
                }
        }
        return min_index; //return index of vertex which is not visited and having
minimum cost

}
int PhoneGraph::prim()
{
        bool visited[MAX];
```

```
int parents[MAX]; //storing vertices
int cost[MAX]; //saving minimum cost
for(int i=0;i<n;i++)
{
        cost[i]=9999;  //set cost as infinity/MAX_VALUE
        visited[i]=0; //initialize visited array to false
}

cost[0]=0; //starting vertex cost
parents[0]=-1; //make first vertex as a root

for(int i=0;i<n-1;i++)
{
        int k=mincost(cost,visited); //minimum cost elemets index
        visited[k]=1; //set visited

        for(int j=0;j<n;j++)//for adjacent verices comparision
        {
                if(data[k][j] && visited[j]==0 && data[k][j] < cost[j])
                {
                        parents[j]=k;
                        cost[j]=data[k][j];
                }
        }
}
```

```
        cout<<"Minimum Cost Telephone Map:\n";
        for(int i=1;i<n;i++)
        {
                cout<<i<<" -- "<<parents[i]<<" = "<<cost[i]<<endl;
        }
        int mincost=0;
        for (int i = 1; i < n; i++)
                mincost+=cost[i];                    //data[i][parents[i]];
        return mincost;
}
//------- Kruskal's Algorithm
void PhoneGraph::kruskal(EdgeList &spanlist)
{
        int belongs[MAX]; //Separate Components at start (No Edges, Only vertices)
        int cno1,cno2;       //Component 1 & 2
        EdgeList elist;
        for(int i=1;i<n;i++)
                for(int j=0;j<i;j++)
                {
                        if(data[i][j]!=0)
                        {
                                elist.data[elist.n]=Edge(i,j,data[i][j]); //constructor for
initializing edge
                                elist.n++;
                        }
```

```
        }
    elist.sort(); //sorting in increasing weight order
    for(int i=0;i<n;i++)
        belongs[i]=i;

    for(int i=0;i<elist.n;i++)
    {
        cno1=find(belongs,elist.data[i].u); //find set of u
        cno2=find(belongs,elist.data[i].v); ////find set of v
        if(cno1!=cno2) //if u & v belongs to different sets
        {
            spanlist.data[spanlist.n]=elist.data[i]; //ADD Edge to spanlist
            spanlist.n=spanlist.n+1;
            unionComp(belongs,cno1,cno2); //ADD both components to same
set
        }
    }
}
void PhoneGraph::unionComp(int belongs[],int c1,int c2)
{
    for(int i=0;i<n;i++)
    {
        if(belongs[i]==c2)
            belongs[i]=c1;
    }
```

```
}
int PhoneGraph::find(int belongs[],int vertexno)
{
	return belongs[vertexno];
}

//--------- MAIN PROGRAM----------------------------------
int main() {
	int vertices,choice;
	EdgeList spantree;
	cout<<"Enter Number of cities: ";
	cin>>vertices;
	PhoneGraph p1(vertices);
	//p1.readgraph();
	do
	{
		cout<<"\n1.Find Minimum Total Cost(By Prim's Algorithm)"
				<<"\n2.Find Minimum Total Cost(by Kruskal's Algorithms)"
				<<"\n3.Re-Read Graph(INPUT)"
				<<"\n4.Print Graph"
				<<"\n0. Exit"
				<<"\nEnter your choice: ";
		cin>>choice;
		switch(choice)
		{
```

```
		case 1:
			cout<<" Minimum cost of Phone Line to cities is: "<<p1.prim();
			break;
		case 2:
			p1.kruskal(spantree);
			spantree.print();
			break;
		case 3:
			p1.readgraph();
			break;
		case 4:
			p1.printGraph();
			break;
		default:
			cout<<"\nWrong Choice!!!";
		}
	}while(choice!=0);

	return 0;
}
/*	Sample INPUT: vertices =7
 *		{{0, 28, 0, 0, 0,10,0},
	 {28,0,16,0,0,0,14},
	 {0,16,0,12,0,0,0},
	 {0,0,12,0,22,0,18},
```

{0,0,0,22,0,25,24},

{10,0,0,0,25,0,0},

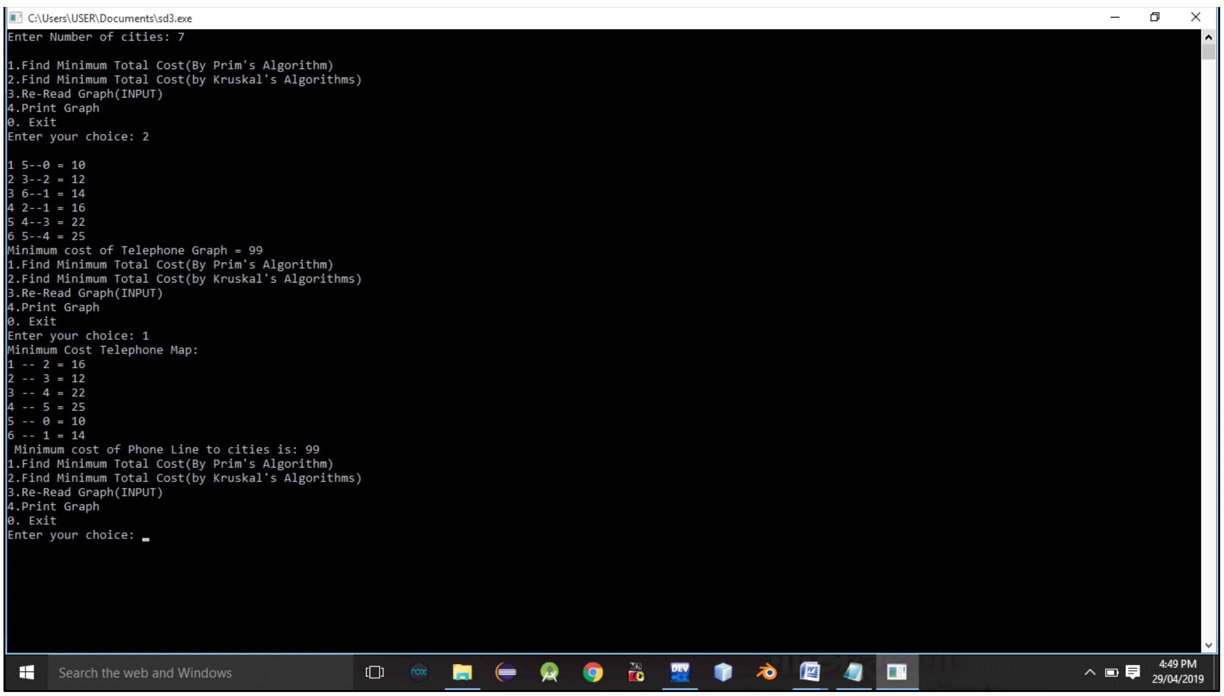
{0,14,0,18,24,0,0},

};

Minimum Cost: 99

*/

Output Screenshots:-



Conclusion:- Thus,we have studied implementation of kruskal's algorithm.

ASSIGNMENT NO.6.

Aim :-

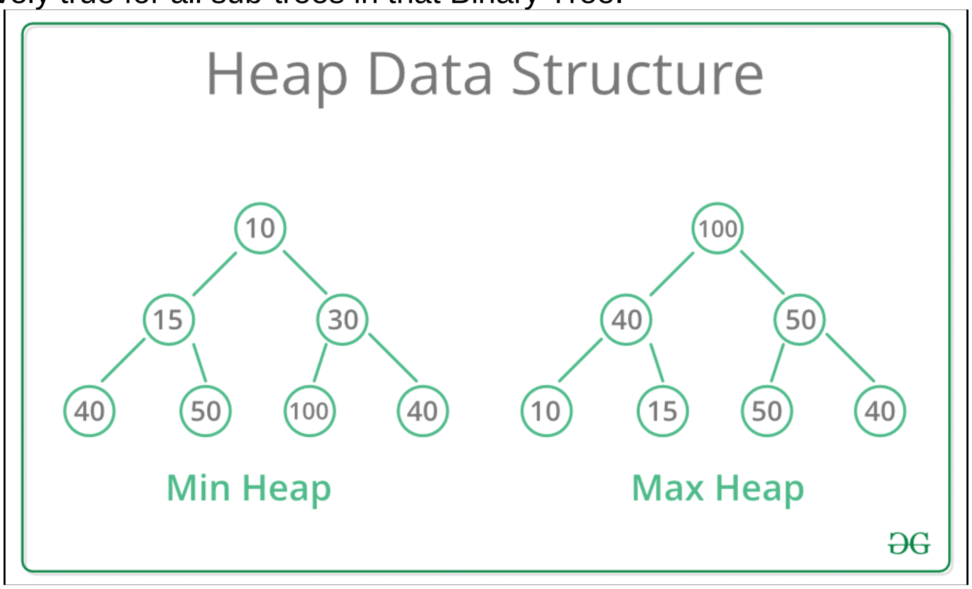
Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject using heap data structure.

Objective:- To study the heap data structure.

Theory:-

A Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types:

- **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
- **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.



Applications:-

1) Heap Sort: Heap Sort uses Binary Heap to sort an array in O(nLogn) time.

2) Priority Queue: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in

O(logn) time. Binomoial Heap and Fibonacci Heap are variations of Binary Heap. These variations perform union also efficiently.
3) Graph Algorithms: The priority queues are especially used in Graph Algorithms like Dijkstra's Shortest Path and Prim's Minimum Spanning Tree.
4) Many problems can be efficiently solved using Heaps. See following for example.
a) K'th Largest Element in an array.
b) Sort an almost sorted array/
c) Merge K Sorted Arrays.

Algorithm:-

1.max heap:-

Step 1 – Create a new node at the end of heap.
Step 2 – Assign new value to the node.
Step 3 – Compare the value of this child node with its parent.
Step 4 – If value of parent is less than child, then swap them.
Step 5 – Repeat step 3 & 4 until Heap property holds.

Program Code:-

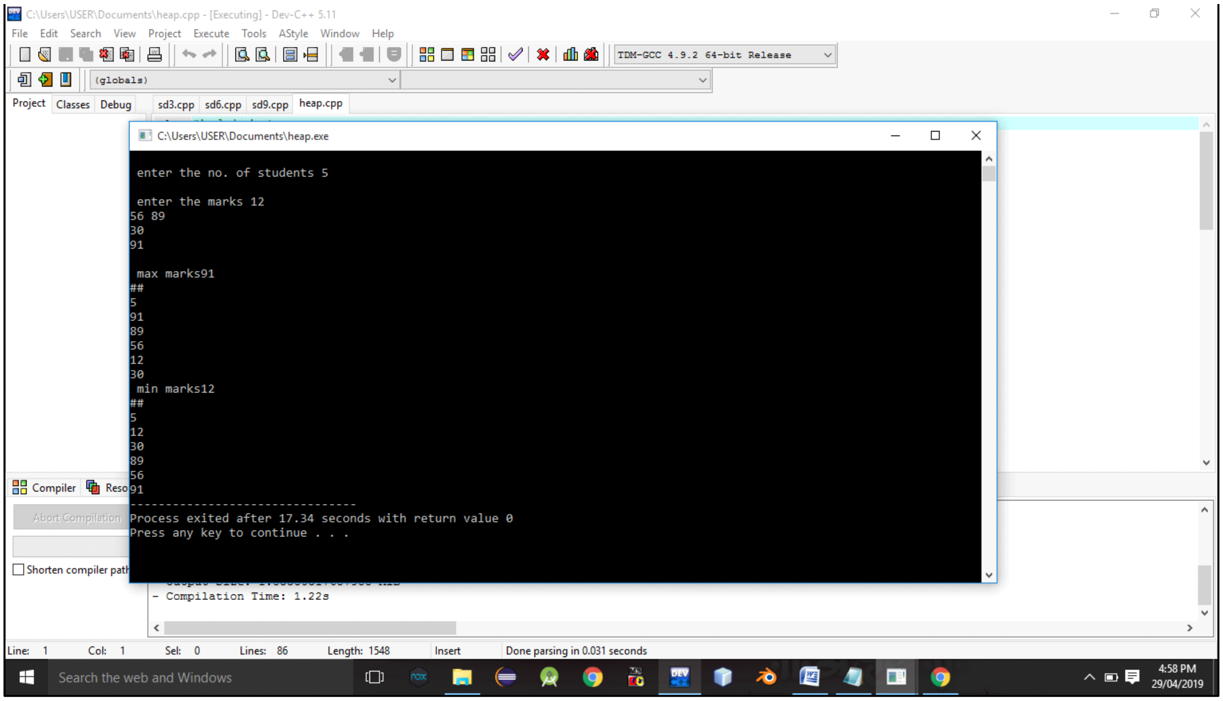
```
#include<iostream>

using namespace std;


class hp

{

  int heap[20],heap1[20],x,n1,i;

  public:

  hp()

  { heap[0]=0;  heap1[0]=0;

  }

  void getdata();

  void insert1(int heap[],int);

  void upadjust1(int heap[],int);

  void insert2(int heap1[],int);

  void upadjust2(int heap1[],int);

  void minmax();
```

```
};
void hp::getdata()
{
   cout<<"\n enter the no. of students";
   cin>>n1;
   cout<<"\n enter the marks";
   for(i=0;i<n1;i++)
   {   cin>>x;
       insert1(heap,x);
       insert2(heap1,x);
   }
}
void hp::insert1(int heap[20],int x)
{
   int n;
   n=heap[0];
   heap[n+1]=x;
   heap[0]=n+1;

   upadjust1(heap,n+1);
}
void hp::upadjust1(int heap[20],int i)
{
    int temp;
    while(i>1&&heap[i]>heap[i/2])
    {
       temp=heap[i];
```

```
        heap[i]=heap[i/2];
        heap[i/2]=temp;
        i=i/2;
    }
}
void hp::insert2(int heap1[20],int x)
{
   int n;
   n=heap1[0];
   heap1[n+1]=x;
   heap1[0]=n+1;

   upadjust2(heap1,n+1);
}
void hp::upadjust2(int heap1[20],int i)
{
    int temp1;
    while(i>1&&heap1[i]<heap1[i/2])
    {
        temp1=heap1[i];
        heap1[i]=heap1[i/2];
        heap1[i/2]=temp1;
        i=i/2;
    }
}
void hp::minmax()
{
```

```
   cout<<"\n max marks"<<heap[1];
   cout<<"\n##";
   for(i=0;i<=n1;i++)
   {   cout<<"\n"<<heap[i];  }
   cout<<"\n min marks"<<heap1[1];
   cout<<"\n##";
   for(i=0;i<=n1;i++)
   {   cout<<"\n"<<heap1[i];  }
}
int main()
{
  hp h;
  h.getdata();
  h.minmax();
  return 0;
}
```

Output Screenshots:-



Conclusion:- Thus,we have studied heap data structure,

ASSIGNMENT NO.7.

Aim :- Insert the keys into a hash table of length m using open addressing using double hashing with h(k)=1+(k mod(m-1)).

Objective:- to study the hashing concept and its techniques.

Theory:-

Open Addressing—

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).
Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete(k): ***Delete operation is interesting***. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".
Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Open Addressing is done following ways:

a) Linear Probing: In linear probing, we linearly probe for next slot. For example, typical gap between two probes is 1 as taken in below example also.
let **hash(x)** be the slot index computed using hash function and **S** be the table size

```
If slot hash(x) % S is full, then we try (hash(x) + 1) % S

If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S

If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S
```

Double Hashing:-

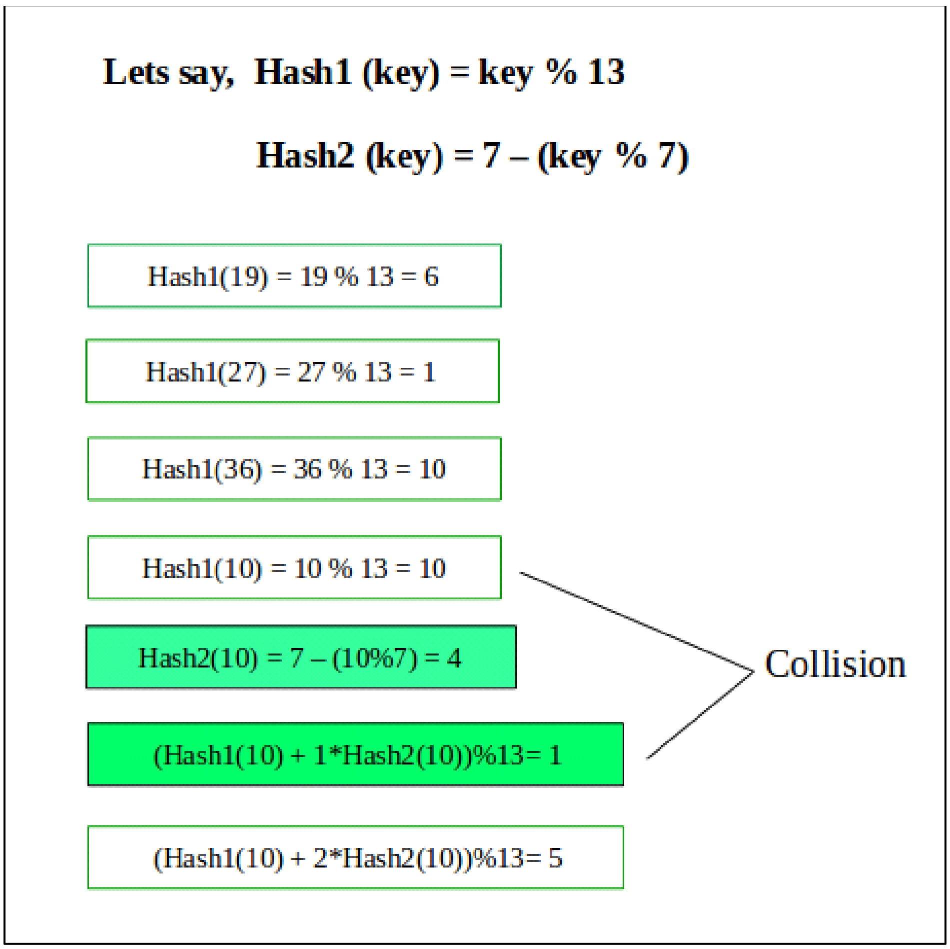
Double hashing is a collision resolving technique in **Open Addressed** HYPERLINK "https://www.geeksforgeeks.org/hashing-set-3-open-addressing/"HYPERLINK "https://www.geeksforgeeks.org/hashing-set-3-open-addressing/" HYPERLINK "https://www.geeksforgeeks.org/hashing-set-3-open-addressing/" Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

Double hashing can be done using :
(hash1(key) + i * hash2(key)) % TABLE_SIZE
Here hash1() and hash2() are hash functions and TABLE_SIZE is size of hash table.
(We repeat by increasing i when collision occurs)

First hash function is typically hash1(key) = key % TABLE_SIZE

A popular second hash function is : **hash2(key) = PRIME – (key % PRIME)** where PRIME is a prime smaller than the TABLE_SIZE.
A good second Hash function is:

- It must never evaluate to zero
- Must make sure that all cells can be probed



Algorithm:-

Program Code:-

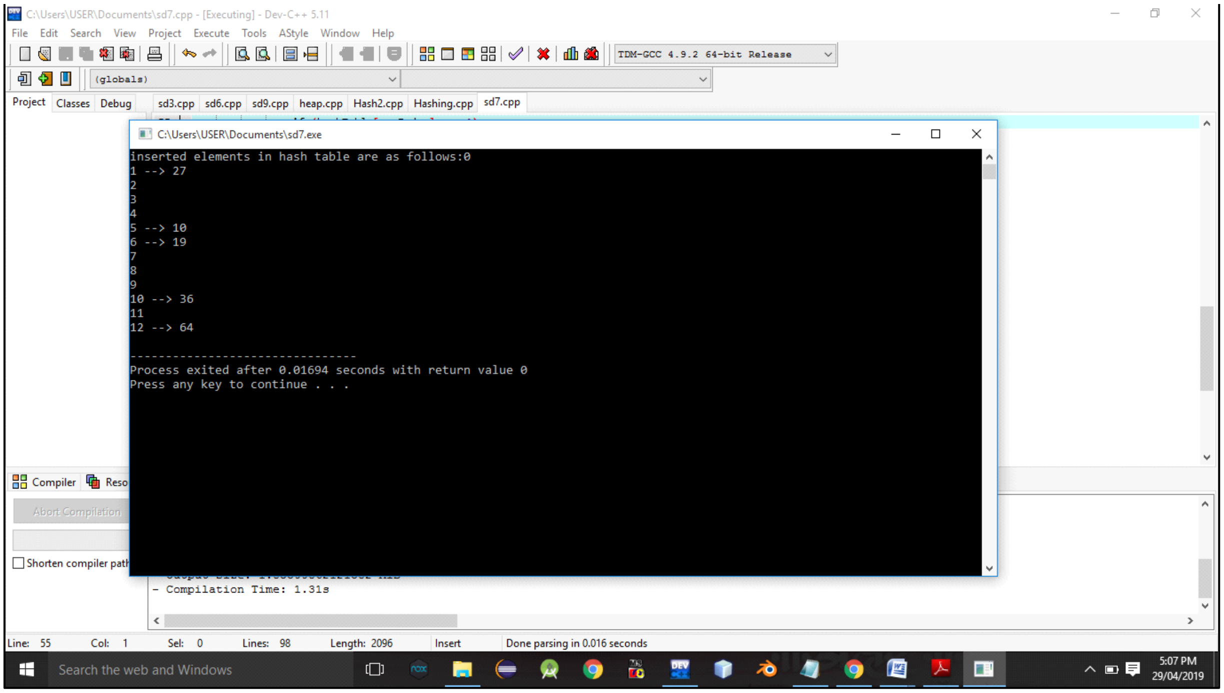
```
#include <bits/stdc++.h>
using namespace std;

#define TABLE_SIZE 13

#define PRIME 7

class DoubleHash
{
    int *hashTable;
    int curr_size;

public:

    bool isFull()
    {

        return (curr_size == TABLE_SIZE);
    }
    int hash1(int key)
    {
        return (key % TABLE_SIZE);
    }
```

```
int hash2(int key)
{
    return (PRIME - (key % PRIME));
}

DoubleHash()
{
    hashTable = new int[TABLE_SIZE];
    curr_size = 0;
    for (int i=0; i<TABLE_SIZE; i++)
        hashTable[i] = -1;
}
void insertHash(int key)
{
    if (isFull())
        return;

    int index = hash1(key);

    if (hashTable[index] != -1)
    {
        int index2 = hash2(key);
        int i = 1;
        while (1)
```

```
        {
            int newIndex = (index + i * index2) %
                                    TABLE_SIZE;

            if (hashTable[newIndex] == -1)
            {
                hashTable[newIndex] = key;
                break;
            }
            i++;
        }
    }

    else
        hashTable[index] = key;
    curr_size++;
}

// function to display the hash table
void displayHash()
{
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        if (hashTable[i] != -1)
            cout << i << " --> "
```

```
                    << hashTable[i] << endl;
            else
                cout << i << endl;
        }
    }
};

// Driver's code
int main()
{
    int a[] = {19, 27, 36, 10, 64};
    int n = sizeof(a)/sizeof(a[0]);
  cout<<"inserted elements in hash table are as follows:";
    // inserting keys into hash table
    DoubleHash h;
    for (int i = 0; i < n; i++)
        h.insertHash(a[i]);

    // display the hash Table
    h.displayHash();
    return 0;
}
```

Output Screenshots:-



<u>Conclusion:-</u> Thus,we have studied double hashing and hashing technique.

ASSIGNMENT: 8

AIM:

Department maintains a student information. The file contains roll number, name, division and address.

Allow user to add, delete information of student. Display information of particular employee. If record of

student does not exist an appropriate message is displayed. If it is, then the system displays the student details Use Sequential file to maintain data

Objective :

To emplement sequential file in c++.

Theory :

Storing and sorting in contiguous block within files on tape or disk is called as sequential access file organization.

In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.

Sequential file search starts from the beginning of the file and the records can be added at the end of the file.

In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.

Advantages of sequential file

It is simple to program and easy to design.

Sequential file is best use if storage space.

Disadvantages of sequential file

Sequential file is time consuming process.

It has high data redundancy.

Random searching is not possible.

Algorithm for Sequential File Allocation:
Step 1: Start the program.
Step 2: Get the number of memory partition and their sizes.
Step 3: Get the number of processes and values of block size for each process.
Step 4: First fit algorithm searches all the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.
Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates it.
Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.
Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.
Step 8: Stop the program.

CODE:

```
#include<iostream>

#include<fstream>

using namespace std;

class student

{

	int roll_num;

	char div;

	string name;

	string address;

	public:

	void getdata()
```

```
{
        cout<<"\n Enter the Roll Number: ";
        cin>>roll_num;
        cout<<"Enter the division: ";
        cin>>div;
        cout<<"Enter the Name: ";
        fflush(stdin);
        getline(cin,name);
        cout<<"Enter the Address: ";
        fflush(stdin);
        getline(cin,address);
}
void putdata(int n)
{
        student st[n];
        ifstream infile;
        infile.open("student.dat",ios::binary|ios::in);
        for(int i=0;i<n;i++)
    {
                infile.read((char *)&st[i],sizeof(st[i]));
                cout<<"\n Roll Number: "<<st[i].roll_num;
                cout<<"\n Division:  "<<st[i].div;
                fflush(stdin);
```

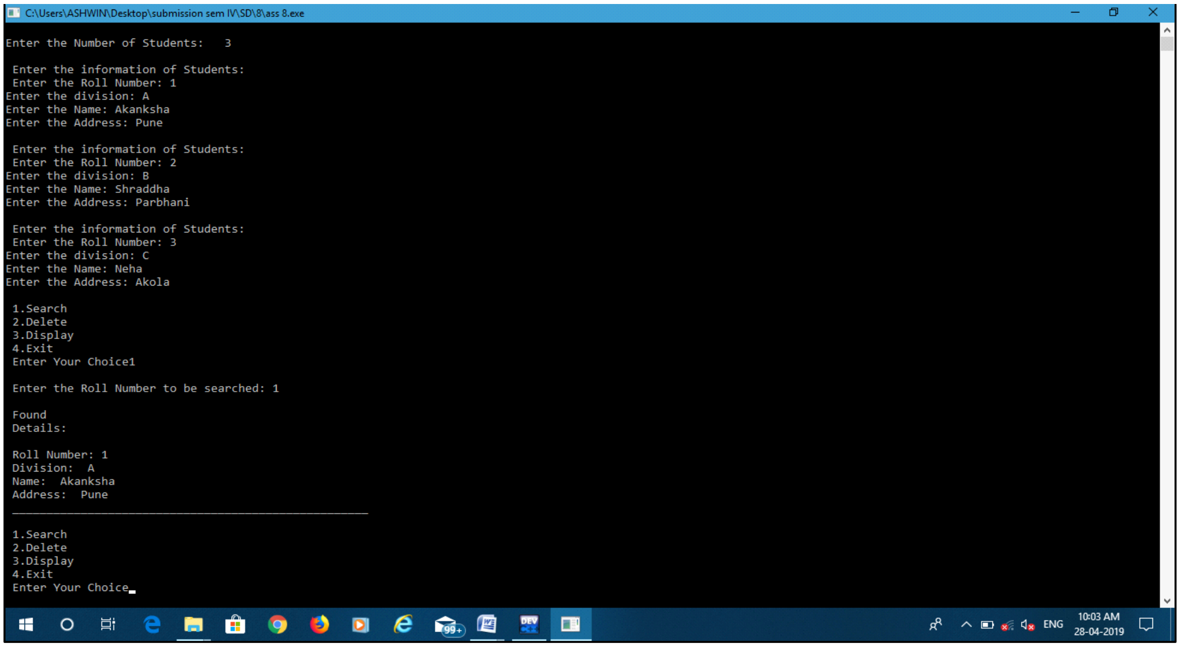
```
                    cout<<"\n Name:  "<<st[i].name;
                    fflush(stdin);
                    cout<<"\n Address:  "<<st[i].address;
                    cout<<"\n
________________________________________________________ \n";
            }
                infile.close();
    }



        void search_(int n)
        {
                student st[n];
                ifstream infile;
                cout<<"\n Enter the Roll Number to be searched: ";
                int r;
                cin>>r;
                infile.open("student.dat",ios::in|ios::binary);
                for(int i=0;i<n;i++)
                {
                    infile.read((char *)&st[i],sizeof(st[i]));
                    if(st[i].roll_num==r)
                    {
```

```
                cout<<"\n Found";
                cout<<"\n Details:  "<<endl;
                cout<<"\n Roll Number: "<<st[i].roll_num;
                cout<<"\n Division:  "<<st[i].div;
                fflush(stdin);
                cout<<"\n Name:  "<<st[i].name;
                fflush(stdin);
                cout<<"\n Address:  "<<st[i].address;
    cout<<"\n_____________________________________________________
\n";
                infile.close();
                return;
            }
        }
        cout<<"\n Not Found";
        infile.close();
    }

    void del(int n)
    {
        student st[n];
        int r;
        cout<<"\n Enter the roll number to be deleted ";
```

```
cin>>r;
ifstream infile;
ofstream outfile;
infile.open("student.dat",ios::binary|ios::in);
outfile.open("temp.dat",ios::binary|ios::out);
for(int i=0;i<n;i++)
{
	infile.read((char *)&st[i],sizeof(st[i]));
	if(st[i].roll_num==r)
	{
		continue;
	}
	else
	{
		outfile.write((char *)&st[i],sizeof(st[i]));
	}
}
outfile.close();
infile.close();
remove("student.dat");
int re=rename("temp.dat","student.dat");
cout<<"Data deleted";
```

```
        }
};
int main()
{
        int n;
        cout<<"\nEnter the Number of Students:   ";
        cin>>n;
        student s[n];
        ofstream outfile;
        outfile.open("student.dat",ios::out|ios::binary);
        for(int i=0;i<n;i++)
        {
                cout<<"\n Enter the information of Students: ";
                s[i].getdata();
                outfile.write((char *)&s[i],sizeof(s[i]));
        }
        outfile.close();

        int c;
        student d;
        do
        {
                cout<<"\n 1.Search";
```

```
        cout<<"\n 2.Delete";
        cout<<"\n 3.Display";
        cout<<"\n 4.Exit";
        cout<<"\n Enter Your Choice";
        cin>>c;
        switch(c)
        {
            case 1:d.search_(n);break;
            case 2:d.del(n);n=n-1;break;
            case 3:d.putdata(n);break;
            case 4:break;
        }
    }
    while(c!=4);

}
```



Conclusion:

Hence Sequential file is implemented for employee data storage.

ASSIGNMENT NO.9.

Aim :- Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

Objective:- to study use of different data structure concepts in this program.

Theory:-

Input/output formatting

Writing to or reading from a file is similar to writing onto a terminal screen or reading from a keyboard. Differences are:

- File must be opened with an OPEN statement, in which the unit number and (optionally) the filename are given
- Subsequent writes (or reads) must refer to a known unit number (used for open)
- File should be closed at the end

File opening and closing

The syntax is:

```
OPEN([unit=]lunit,file='name' [,options])

CLOSE([unit=]lunit [,options])
```

For example:

```
OPEN(10, file='output.dat', status='new')

CLOSE(unit=10)
```

- The first parameter is the unit number and the keyword unit= can be omitted.
- The unit numbers 0,5 and 6 are predefined.
- 0 is output for standard (system) error messages
- 5 is for standard (user) input
- 6 is for standard (user) output

- These units are opened by default and should not be re-opened nor closed by users

Some options for opening a file:

- status: existence of the file ('old', 'new', 'replace', 'scratch', 'unknown')
- position: offset, where to start writing ('append')
- action: file operation mode ('write','read','readwrite')
- form: text or binary file ('formatted', 'unformatted')
- access: direct or sequential file access ('direct','sequential','stream')
- iostat: error indicator, (output) integer (non zero only upon an error)
- err: the label number to jump upon error
- recl: record length, (input) integer for direct access files only. Be careful, it can be in bytes or words...

Algorithm:-

Program Code:-

```
#include <iostream>

#include <fstream>

#include <cstring>

#include <iomanip>

#include<cstdlib>

#define max 50

using namespace std;

class Employee

{

   char name[max];

   int empid;

   int sal;

   char de[50];

   friend class FileOperations;
```

```
public:        Employee()
               {
                              strcpy(name,"");
                              empid=sal==0;
                              strcpy(de,"");
               }
               Employee(char name[max],int empid,int sal,char de[max])
               {
                       strcpy(this->de,de);
                       strcpy(this->name,name);

                       this->empid=empid;
                       this->sal=sal;
               }
               int getEmpId()
               {
                       return empid;
               }
               void displayEmployeeData()
               {

                       cout<<endl<<empid<<"\t\t\t"<<name<<"\t\t\t"<<sal<<"\t\t\t"<<de;
               }
};
```

```
class FileOperations
{
    fstream file;
    public:FileOperations(char *name)
        {
            //strcpy(this->name,name);
            this->file.open(name,ios::in|ios::out|ios::ate|ios::binary);
        }
        void insertRecord(int empid,char name[max],int sal,char de[max])
        {
             Employee s=Employee(name,empid,sal,de);
             file.seekp(0,ios::end);
             file.write((char*)&s,sizeof(Employee));
             file.clear();
        }
        void displayAllRecords()
        {
             Employee s;
             file.seekg(0,ios::beg);
             while(file.read((char *)&s,sizeof(Employee)))
             {
                  s.displayEmployeeData();
             }
             file.clear();
```

```
}
void displayRecord(int empid)
{
    Employee s;
    file.seekg(0,ios::beg);
    void *p;
    while(file.read((char *)&s,sizeof(Employee)))
    {
        if(s.empid==empid)
        {
            s.displayEmployeeData();
            break;
        }
    }
    if(p==NULL)
        throw "Element not present";
    file.clear();
}
void deleteRecord(int empid)
{
    ofstream newFile("new.txt",ios::binary);
    file.seekg(0,ios::beg);
    bool flag=false;
    Employee s;
```

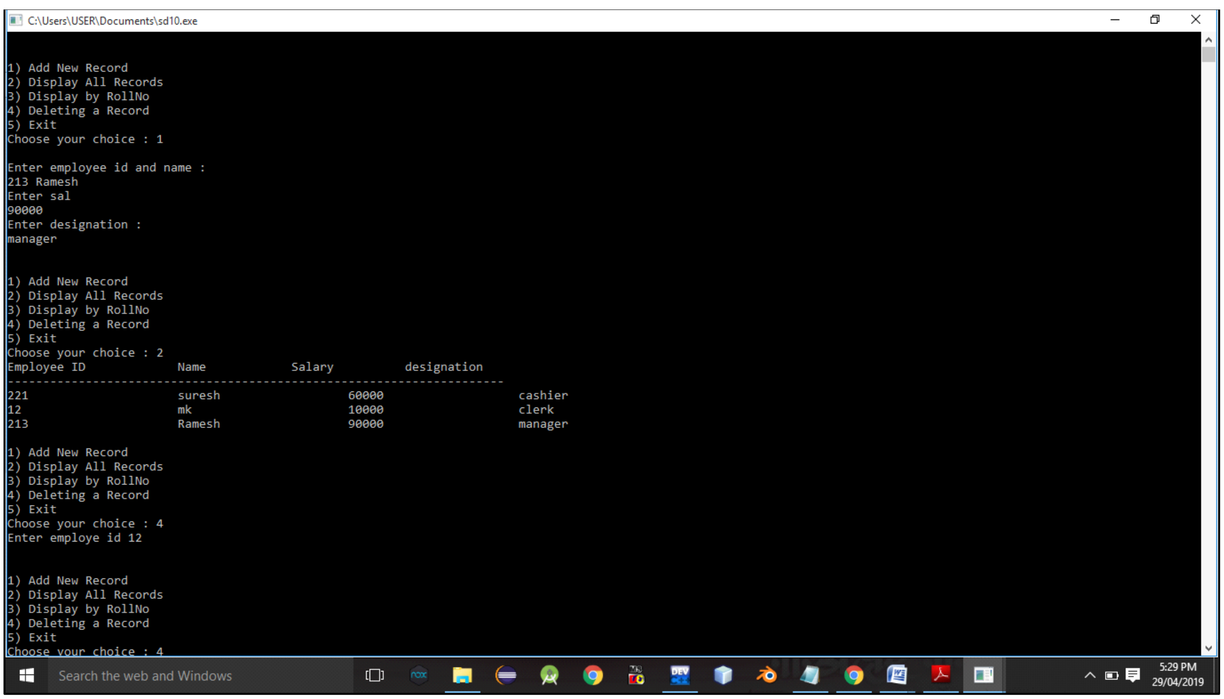
```
        while(file.read((char *)&s,sizeof(s)))
        {
            if(s.empid==empid)
            {
                flag=true;
                continue;
            }
            newFile.write((char *)&s,sizeof(s));
        }
        if(!flag)
        {
            cout<<"Element Not Present";
        }
        file.close();
        newFile.close();
        remove("Employee.txt");
        rename("new.txt","Employee.txt");
        file.open("Employee.txt",ios::in|ios::ate|ios::out|ios::binary);
    }
    ~FileOperations()
    {
        file.close();
        cout<<"Closing file..";
    }
```

```
};
int  main()
{
        ofstream newFile("Employee.txt",ios::app|ios::binary);
        newFile.close();
        FileOperations file((char *)"Employee.txt");
  int empid,sal,choice=0;
  char name[max],de[max];
  while(choice!=5)
  {
    cout<<"\n\n1) Add New Record\n";
    cout<<"2) Display All Records\n";
    cout<<"3) Display by RollNo\n";
    cout<<"4) Deleting a Record\n";
    cout<<"5) Exit\n";
    cout<<"Choose your choice : ";
    cin>>choice;
    switch(choice)
    {
      case 1 : //New Record
               cout<<endl<<"Enter employee id and name : \n";
               cin>>empid>>name;
               cout<<"Enter sal  \n";
```

```
            cin>>sal;
            cout<<"Enter designation : \n";
            cin>>de;
            file.insertRecord(empid,name,sal,de);
            break;
        case 2 :

                    cout<<"Employee
ID"<<"\t\t"<<"Name"<<"\t\t"<<"Salary"<<"\t\t"<<"designation\n";
         cout<<"-------------------------------------------------------------------";
                                file.displayAllRecords();
            break;
        case 3 :
            cout<<"Enter employee id";
            cin>>empid;
            try
            {
                file.displayRecord(empid);
            }
            catch(const char *str)
            {
                cout<<str;
            }
            break;
```

```
        case 4:
                cout<<"Enter employe id";
                cin>>empid;
                file.deleteRecord(empid);
                break;
        case 5 :break;
      }
   }
}
```

Output Screenshots:-



Conclusion:- Thus, this assignment is completed successfully.