

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.

Algorithm :

1. Define to array to performed the set operation .
2. Then performed add function to add element in set.
3. also performed remove function.
4. Return the size of array to get number of element in set.
5. Return element which is not common in both array
6. Merge the two array.

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;j<=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;

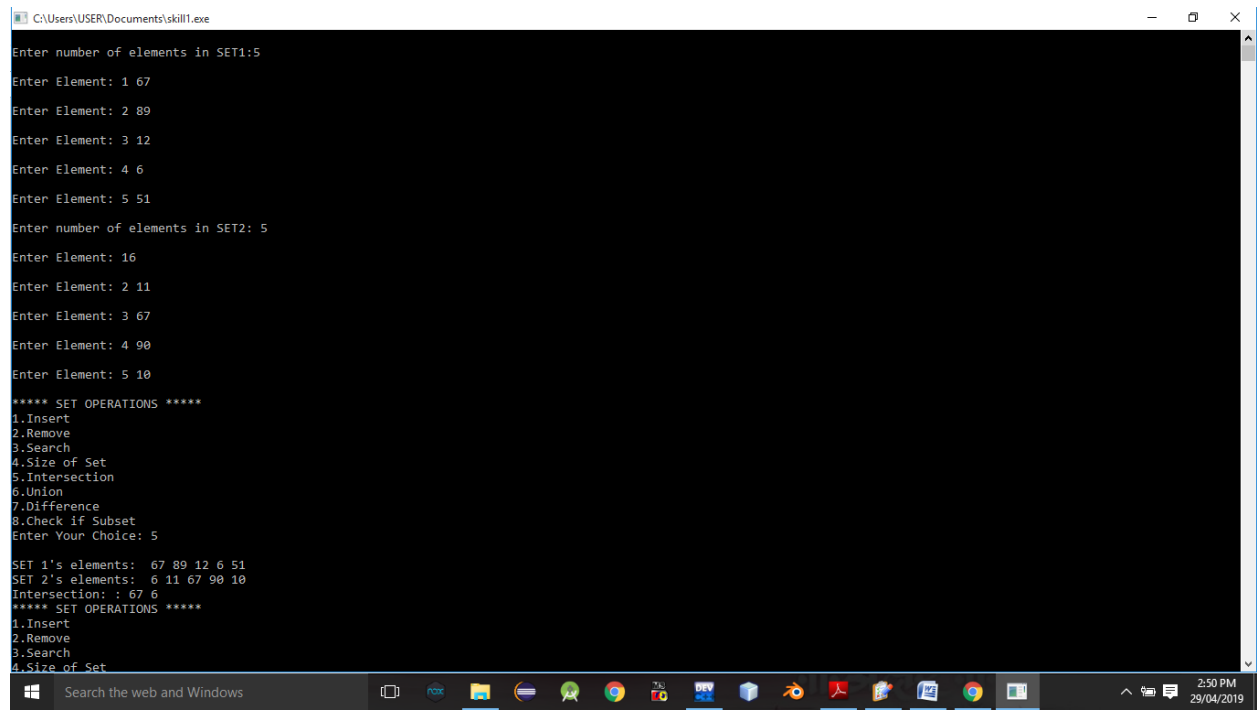
}

}while(choice!=0);

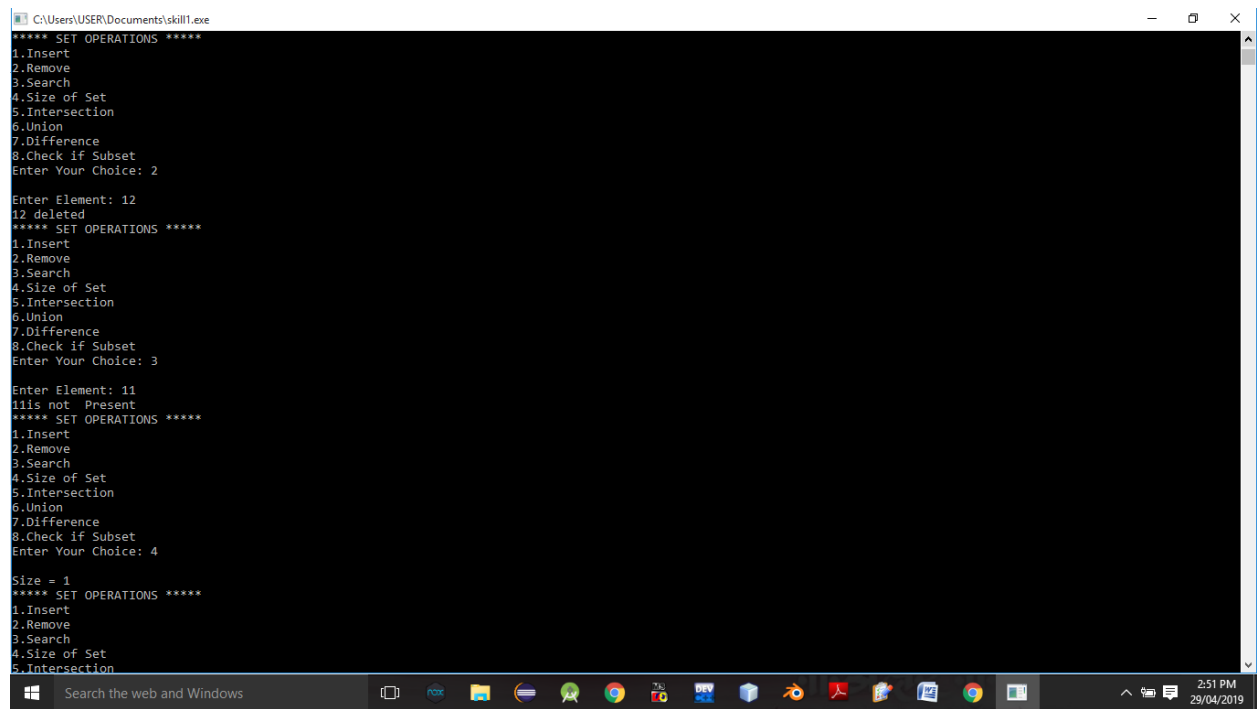
return 0;

}
```

Output Screenshots:-



```
CAUsers\USER\Documents\skill1.exe
Enter number of elements in SET1:5
Enter Element: 1 67
Enter Element: 2 89
Enter Element: 3 12
Enter Element: 4 6
Enter Element: 5 51
Enter number of elements in SET2: 5
Enter Element: 16
Enter Element: 2 11
Enter Element: 3 67
Enter Element: 4 90
Enter Element: 5 10
***** 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: 5
SET 1's elements: 67 89 12 6 51
SET 2's elements: 6 11 67 90 10
Intersection: : 67 6
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
```



```
***** 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: 2
Enter Element: 12
12 deleted
***** 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: 3
Enter Element: 11
11 is not Present
***** 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: 4
Size = 1
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
```

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

ASSIGNMENT 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 understand the concept of binary threading and to understand insertion and inorder traversal on threaded binary tree.

Theory:

Threaded binary tree : A binary tree is *threaded* by making all right child pointers that would normally be null point to the inorder successor of the node (if it exists), and all left child pointers that would normally be null point to the inorder predecessor of the node.

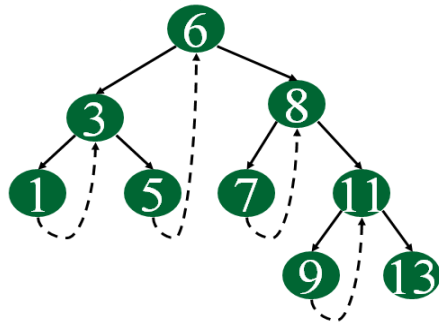
There are two types of threaded binary trees.

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 :

1. Define required structure for threaded binary tree.

```
struct Node
{
    struct Node *left, *right;
    int info;

    // True if left pointer points to predecessor
    // in Inorder Traversal
    boolean lthread;

    // True if right pointer points to successor
    // in Inorder Traversal
    boolean rthread;
};
```

2. To perform insertion 3 cases we need to understand

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 = tmp;
```

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;
```


Code:

```
#include<iostream>
using namespace std;

class ttree
{
    private:
        struct thtree
        {
            int left;
            thtree *leftchild;
            int data;
            thtree *rightchild;
            int right;
        }*th_head;

    public:
        ttree();
        void insert(int num);
        void inorder();
};

ttree::ttree()
{
    th_head=NULL;
}

void ttree::insert(int num)
{
    thtree *head=th_head,*p,*z;

    z=new thtree;
    z->left=true;
```

```
z->data=num;
z->right=true;

if(th_head==NULL)
{
    head=new thtree;
    head->left=false;
    head->leftchild=z;
    head->data=-9999;
    head->rightchild=head;
    head->right=false;

    th_head=head;
    z->leftchild=head;
    z->rightchild=head;
}
else
{
    p=head->leftchild;
    while(p!=head)
    {
        if(p->data > num)
        {
            if(p->left!=true)
            p=p->leftchild;
            else
            {
                z->leftchild=p->leftchild;
                p->leftchild=z;

                p->left=false;
                z->right=true;
                z->rightchild=p;
                return;
            }
        }
    }
}
```

```
        else
        {
            if(p->data < num)
            {
                if(p->right!=true)
                p=p->rightchild;
                else
                {
                    z->rightchild=p->rightchild;
                    p->rightchild=z;

                    p->right=false;
                    z->left=true;
                    z->leftchild=p;
                    return;
                }
            }
        }
    }
}

void ttree::inorder()
{
    thtree *a;
    a=th_head->leftchild;
    while(a!=th_head)
    {
        while(a->left==false)

            a=a->leftchild;
            cout<<a->data<<"\t";

        while(a->right==true)
        {
            a=a->rightchild;
```

```
                if(a==th_head)
                    break;

                cout<<a->data<<"\t";
            }
            a=a->rightchild;
        }
    }

int main()
{
    ttree th;
    int n,e;
    cout<<"Enter no. of elements: ";
    cin>>n;
    cout<<"\nEnter elements: ";
    for(int i=0;i<n;i++)
    {
        cin>>e;
        th.insert(e);
    }

    cout<<"\n\n\tThreaded binary tree in inorder: ";
    th.inorder();
}
```

Output:

```
Enter no. of elements: 6
Enter elements: 7 9 12 15 6 11

Threaded binary tree in inorder: 6 7 9 11 12 15
-----
Process exited after 27.63 seconds with return value 0
Press any key to continue . . .
```

Conclusion : We successfully implement threaded binary tree and perform insertion and inorder traversal operation on it.

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 $V \times V$ 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 .

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

    {

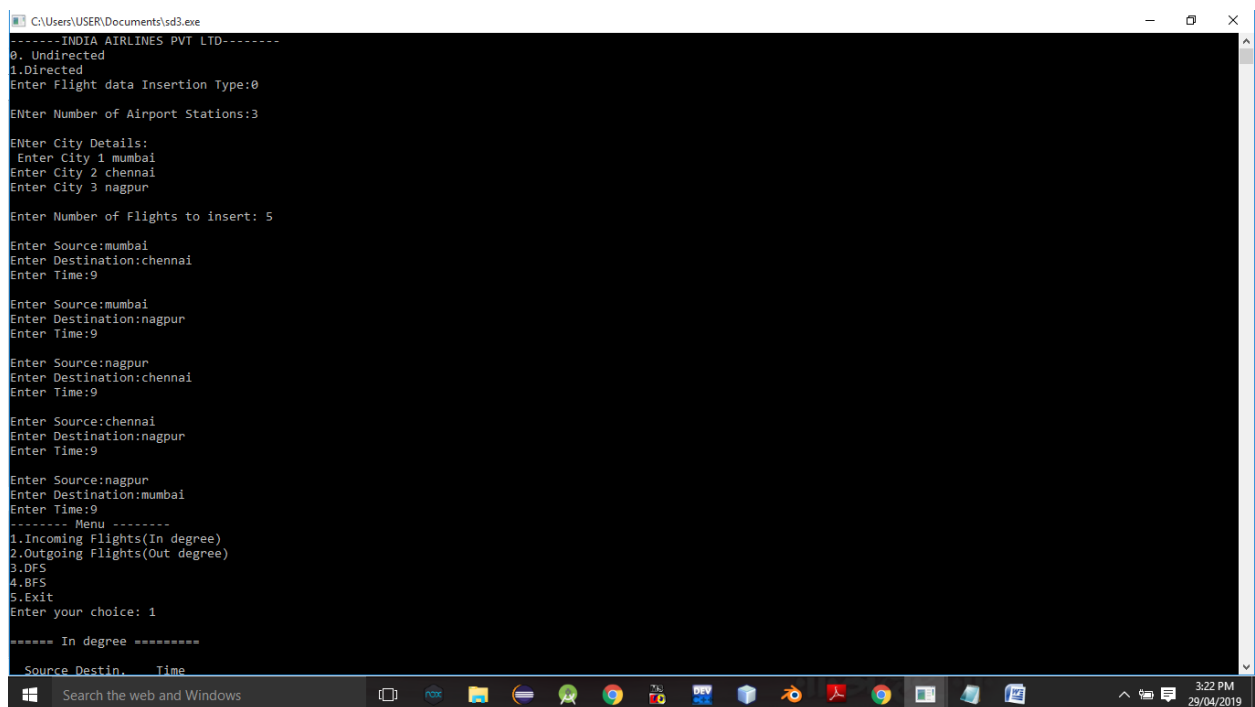
        cout<<"----- Menu -----"
```

```
<<"\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:-



```
C:\Users\USER\Documents\sd3.exe  
-----INDIA AIRLINES PVT LTD-----  
0. Undirected  
1. Directed  
Enter Flight data Insertion Type:0  
  
Enter Number of Airport Stations:3  
  
Enter City Details:  
Enter City 1 mumbai  
Enter City 2 chennai  
Enter City 3 nagpur  
  
Enter Number of Flights to insert: 5  
  
Enter Source:mumbai  
Enter Destination:chennai  
Enter Time:9  
  
Enter Source:mumbai  
Enter Destination:nagpur  
Enter Time:9  
  
Enter Source:nagpur  
Enter Destination:chennai  
Enter Time:9  
  
Enter Source:chennai  
Enter Destination:nagpur  
Enter Time:9  
  
Enter Source:nagpur  
Enter Destination:mumbai  
Enter Time:9  
----- Menu -----  
1.Incoming Flights(In degree)  
2.Outgoing Flights(Out degree)  
3.DFS  
4.BFS  
5.Exit  
Enter your choice: 1  
  
===== In degree =====  
Source Destin. Time
```

```

C:\Users\USER\Documents\sd3.exe
===== In degree =====
Source Destin. Time
chennai mumbai 9
nagpur mumbai 9
nagpur mumbai 9
Flights to mumbai = 3
-----
Source Destin. Time
mumbai chennai 9
nagpur chennai 9
nagpur chennai 9
Flights to chennai = 3
-----
Source Destin. Time
mumbai nagpur 9
mumbai nagpur 9
chennai nagpur 9
chennai nagpur 9
Flights to nagpur = 4
-----
----- Menu -----
1.Incoming Flights(In degree)
2.Outgoing Flights(Out degree)
3.DFS
4.BFS
5.Exit
Enter your choice: 2

Source Destin. Time
mumbai chennai 9
mumbai nagpur 9
mumbai nagpur 9
No. of flights: 3
-----
Source Destin. Time
chennai mumbai 9
chennai nagpur 9
chennai nagpur 9
No. of flights: 3
-----

```

```

C:\Users\USER\Documents\sd3.exe
Source Destin. Time
chennai mumbai 9
chennai nagpur 9
chennai nagpur 9
No. of flights: 3
-----
Source Destin. Time
nagpur mumbai 9
nagpur chennai 9
nagpur chennai 9
nagpur mumbai 9
No. of flights: 4
-----
----- Menu -----
1.Incoming Flights(In degree)
2.Outgoing Flights(Out degree)
3.DFS
4.BFS
5.Exit
Enter your choice: 3
DFS TRAVERSAL:
mumbai chennai nagpur
----- Menu -----
1.Incoming Flights(In degree)
2.Outgoing Flights(Out degree)
3.DFS
4.BFS
5.Exit
Enter your choice: 4
BFS Traversal:
mumbai chennai nagpur
----- Menu -----
1.Incoming Flights(In degree)
2.Outgoing Flights(Out degree)
3.DFS
4.BFS
5.Exit
Enter your choice: 5_

```

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

ASSIGNMENT NO.4.

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

Objective:- To study the prim's 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 vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices)

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 <<"enter no of vertices";
    cin >> n;
    cout <<"enter 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:-

```
CAUsers\USER\Documents\sd3.exe
Enter no of vertices 5
Enter no of edges 6

EDGES Cost
1 2 3
2 3 2
3 4 3
4 5 5
5 1 2
2 4 4
ORDER OF VISITED VERTICES 2 3 3 3
-----
Process exited after 50.36 seconds with return value 0
Press any key to continue . . .
```

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;

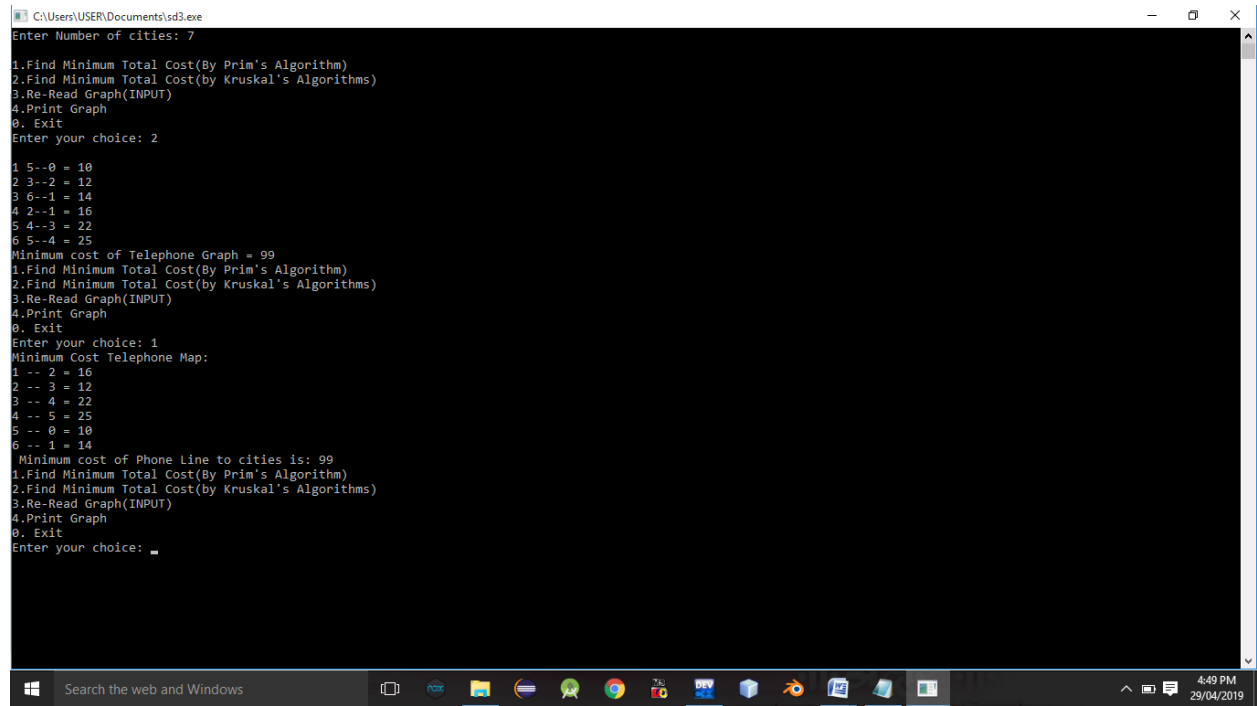
    }

/*   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:-



```
CAUsers\USER\Documents\sd3.exe
Enter Number of cities: 7
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 2
1 5--0 = 10
2 3--2 = 12
3 6--1 = 14
4 2--1 = 16
5 4--3 = 22
6 5--4 = 25
Minimum cost of Telephone Graph = 99
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 1
Minimum Cost Telephone Map:
1 -- 2 = 16
2 -- 3 = 12
3 -- 4 = 22
4 -- 5 = 25
5 -- 0 = 10
6 -- 1 = 14
Minimum cost of Phone Line to cities is: 99
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: _
```

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

ASSIGNMENT NO.6.

Aim :-

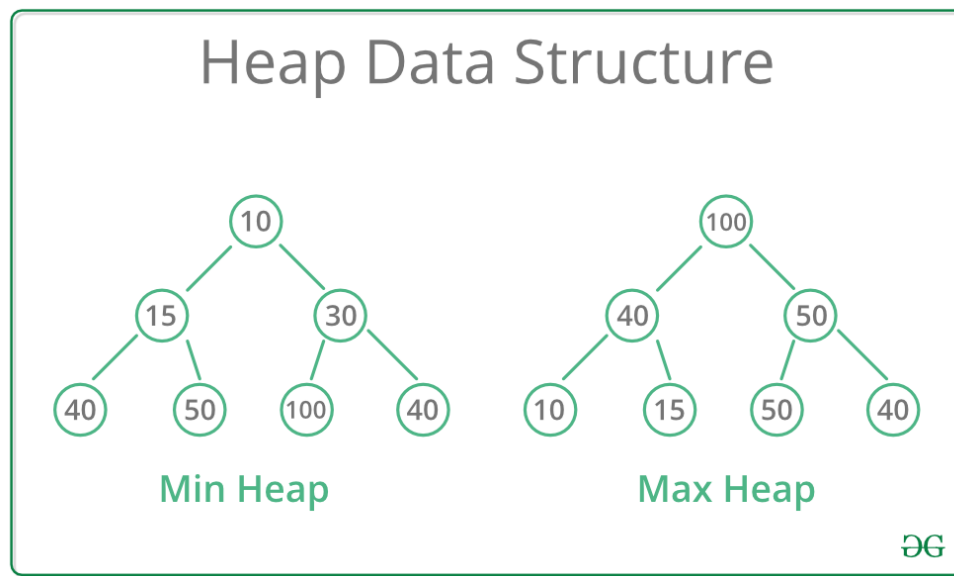
Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject using heap data structure.

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:

1. **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.
2. **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.
- 2.Priority Queue.
- 3.Graph algorithm shortest path e.t.c.

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:-

The screenshot shows the Dev-C++ IDE with a project named 'heap.cpp'. The program is running, and the output window displays the following text:

```
enter the no. of students 5
enter the marks 12
56 89
30
91
max marks91
##
5
91
89
56
12
30
min marks12
##
5
12
30
89
56
91
```

Below the output window, a message box indicates: "Process exited after 17.34 seconds with return value 0. Press any key to continue . . .". The status bar at the bottom shows "Line: 1 Col: 1 Sel: 0 Lines: 86 Length: 1548 Done parsing in 0.031 seconds".

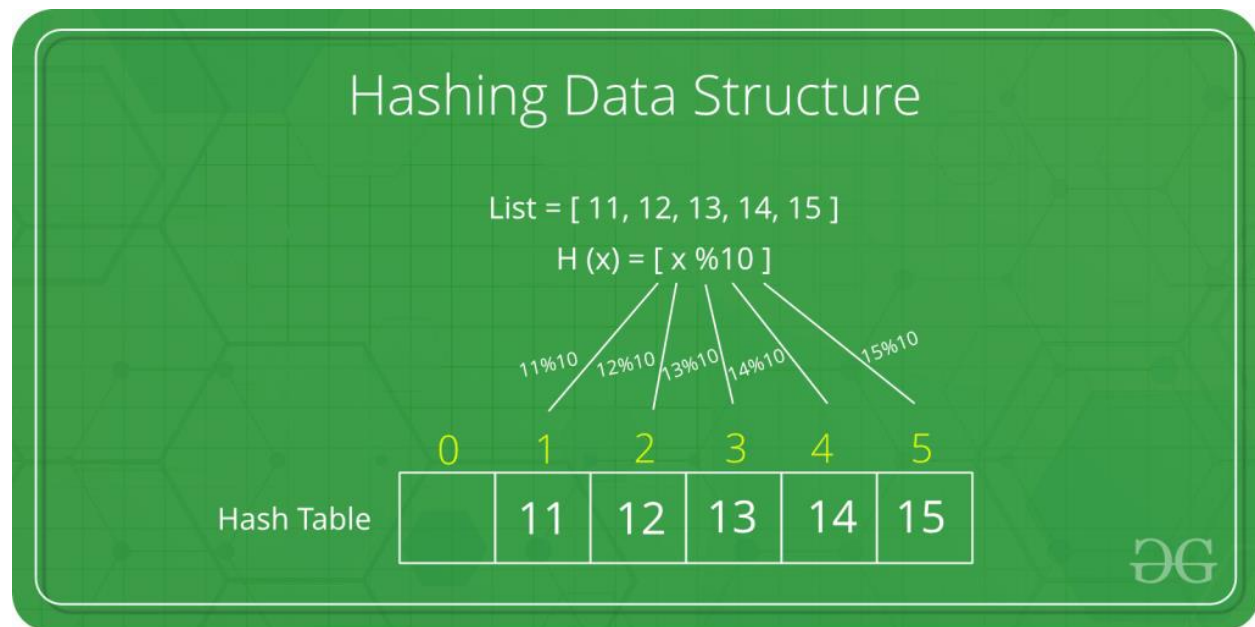
Conclusion:- Thus,we have studied heap data structure,

ASSIGNMENT NO.7.

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

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 $\text{hash}(x) \% S$ is full, then we try $(\text{hash}(x) + 1) \% S$

If $(\text{hash}(x) + 1) \% S$ is also full, then we try $(\text{hash}(x) + 2) \% S$

If $(\text{hash}(x) + 2) \% S$ is also full, then we try $(\text{hash}(x) + 3) \% S$

Double Hashing:-

Double hashing is a collision resolving technique in **Open Addressed** 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 :

$(\text{hash1}(\text{key}) + i * \text{hash2}(\text{key})) \% \text{TABLE_SIZE}$

Here $\text{hash1}()$ and $\text{hash2}()$ are hash functions and TABLE_SIZE is size of hash table.

(We repeat by increasing i when collision occurs)

First hash function is typically $\text{hash1}(\text{key}) = \text{key} \% \text{TABLE_SIZE}$

A popular second hash function is : **$\text{hash2}(\text{key}) = \text{PRIME} - (\text{key} \% \text{PRIME})$** 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

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:-

```

C:\Users\USER\Documents\sd7.exe
Inserted elements in hash table are as follows:0
1 --> 27
2 --> 10
3 --> 19
4 --> 36
5 --> 64
.....
Process exited after 0.01694 seconds with return value 0
Press any key to continue . . .
.....
Compilation Time: 1.31s
.....
Line: 55 Col: 1 Sel: 0 Lines: 98 Length: 2096 Insert Done parsing in 0.016 seconds

```

Conclusion:- Thus, we have studied double hashing and hashing technique.

ASSIGNMENT NO.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

Objective:- To study the different data structure concepts to implement 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...

Program Code:-

```
#include <iostream>
```

```
#include <fstream>
```

```
#include <cstring>
```

```
#include <iomanip>
```

```
#include<cstdlib>
```

```
#define max 50
```

```
using namespace std;
```

```
class Student
```

```
{
```

```
    char name[max];
```

```
    int rollNo;
```

```
    int year;
```

```
    int division;
```

```
char address[50];

friend class FileOperations;

public:    Student()
        {
            strcpy(name,"");
            rollNo=year=division=0;
            strcpy(address,"");
        }

    Student(char name[max],int rollNo,int year,int division,char
address[max])
    {
        strcpy(this->address,address);
        strcpy(this->name,name);
        this->division=division;
        this->rollNo=rollNo;
        this->year=year;
    }

    int getRollNo()
    {
        return rollNo;
    }

    void displayStudentData()
```

```
        {

            cout<<endl<<setw(3)<<rollNo<<setw(10)<<name<<setw(3)<<year<<setw(2)
            )<<division<<setw(10)<<address;

        }

};

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 rollNo,char name[max],int year,int
    division,char address[max])
    {
        Student s=Student(name,rollNo,year,division,address);

        file.seekp(0,ios::end);

        file.write((char*)&s,sizeof(Student));

        file.clear();

    }

    void displayAllRecords()
```

```
{  
  
    Student s;  
  
    file.seekg(0,ios::beg);  
  
    while(file.read((char *)&s,sizeof(Student)))  
    {  
  
        s.displayStudentData();  
  
    }  
  
    file.clear();  
  
}  
  
void displayRecord(int rollNo)  
{  
  
    Student s;  
  
    file.seekg(0,ios::beg);  
  
    void *p;  
  
    while(file.read((char *)&s,sizeof(Student)))  
    {  
  
        if(s.rollNo==rollNo)  
        {  
  
            s.displayStudentData();  
  
            break;  
  
        }  
  
    }  
  
}
```

```
        if(p==NULL)

            throw "Element not present";

        file.clear();

    }

void deleteRecord(int rollNo)

{

    ofstream newFile("new.txt",ios::binary);

    file.seekg(0,ios::beg);

    bool flag=false;

    Student s;

    while(file.read((char *)&s,sizeof(s)))

    {

        if(s.rollNo==rollNo)

        {

            flag=true;

            continue;

        }

        newFile.write((char *)&s,sizeof(s));

    }

    if(!flag)

    {

        cout<<"Element Not Present";

    }

}
```

```
        }

        file.close();

        newFile.close();

        remove("student.txt");

        rename("new.txt","student.txt");

        file.open("student.txt",ios::in|ios::ate|ios::out|ios::binary);

    }

    ~FileOperations()

    {

        file.close();

        cout<<"Closing file..";

    }

};

int main()

{

    ofstream newFile("student.txt",ios::app|ios::binary);

    newFile.close();

    FileOperations file((char *)"student.txt");

    int rollNo,year,choice=0;

    char division;

    char name[max],address[max];
```

```
while(choice!=5)
{
    //clrscr();

    cout<<"\n*****Phone Book*****\n";
    cout<<"1) 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 RollNo and name : \n";
            cin>>rollNo>>name;
            cout<<"Enter Year and Division : \n";
            cin>>year>>division;
            cout<<"Enter address : \n";
            cin>>address;
            file.insertRecord(rollNo,name,year,division,address);
            break;
```


case 2 :

```
file.displayAllRecords();
```

```
break;
```

case 3 :

```
cout<<"Enter Roll Number";
```

```
cin>>rollNo;
```

```
try
```

```
{
```

```
file.displayRecord(rollNo);
```

```
}
```

```
catch(const char *str)
```

```
{
```

```
cout<<str;
```

```
}
```

```
break;
```

case 4:

```
cout<<"Enter rollNo";
```

```
cin>>rollNo;
```

```
file.deleteRecord(rollNo);
```

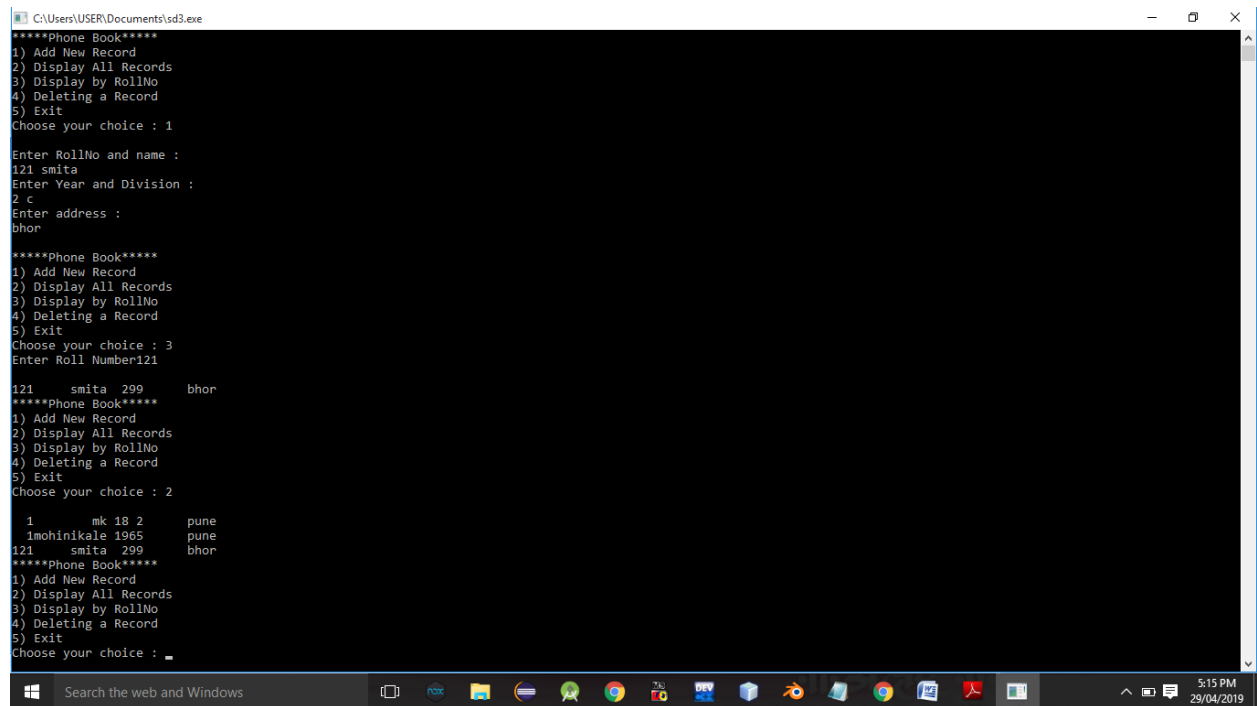
```
break;
```

case 5 :break;

```
}
```

```
}  
  
}
```

Output Screenshots:-



```
CAUsers\USER\Documents\sd3.exe  
*****Phone Book*****  
1) Add New Record  
2) Display All Records  
3) Display by RollNo  
4) Deleting a Record  
5) Exit  
Choose your choice : 1  
  
Enter RollNo and name :  
121 smita  
Enter Year and Division :  
2 c  
Enter address :  
bhor  
  
*****Phone Book*****  
1) Add New Record  
2) Display All Records  
3) Display by RollNo  
4) Deleting a Record  
5) Exit  
Choose your choice : 3  
Enter Roll Number:121  
  
121 smita 299 bhor  
*****Phone Book*****  
1) Add New Record  
2) Display All Records  
3) Display by RollNo  
4) Deleting a Record  
5) Exit  
Choose your choice : 2  
  
1 mk 18 2 pune  
1mohinikale 1965 pune  
121 smita 299 bhor  
*****Phone Book*****  
1) Add New Record  
2) Display All Records  
3) Display by RollNo  
4) Deleting a Record  
5) Exit  
Choose your choice : _
```

```

C:\Users\USER\Documents\sd3.exe
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 3
Enter Roll Number121

121    smita 299    bhor
*****Phone Book*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 2

1      mk 18 2      pune
1mohinikale 1965    pune
121    smita 299    bhor
*****Phone Book*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 4
Enter rollNo1

*****Phone Book*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 2

121    smita 299    bhor
*****Phone Book*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 5

```

Conclusion:- Thus, this assignment implemented successfully.

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...

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:-

Skill Development Lab-2 ,2018-19

Name : Vipul Lodha GR No:21820044 Roll No:223065

```
CAUsers\USER\Documents\sd10.exe
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 1

Enter employee id and name :
213 Ramesh
Enter sal
90000
Enter designation :
manager

1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 2
Employee ID      Name              Salary              designation
-----
221              suresh              60000              cashier
12               mk                  10000              clerk
213              Ramesh              90000              manager

1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 4
Enter employee id 12

1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 4
```

Conclusion:- Thus, this assignment is completed successfully.