**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](https://brilliant.org/wiki/abstract-data-types/) that allows you to store a list of non-repeated values. Their name derives from the mathematical concept of finite [sets](https://brilliant.org/wiki/sets/).

Unlike an [array](https://brilliant.org/wiki/arrays/), 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;

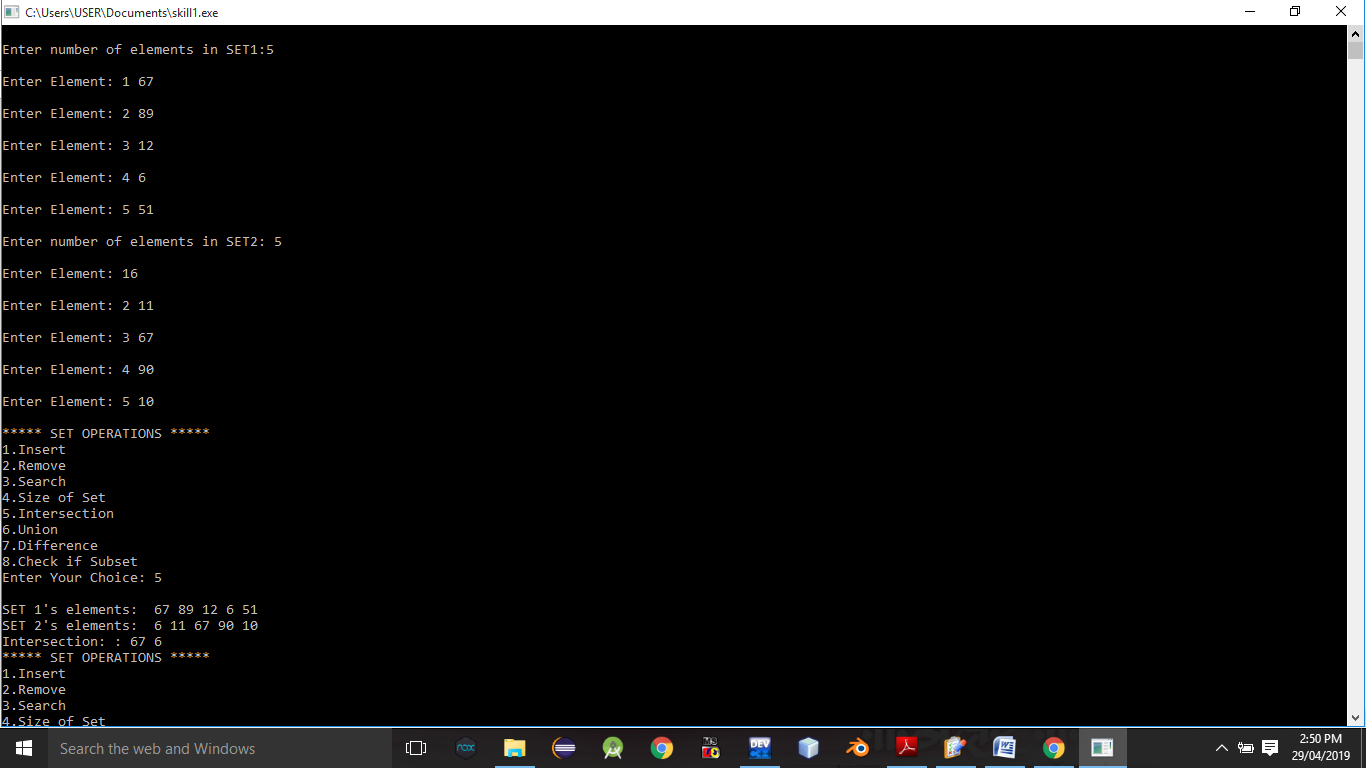
}

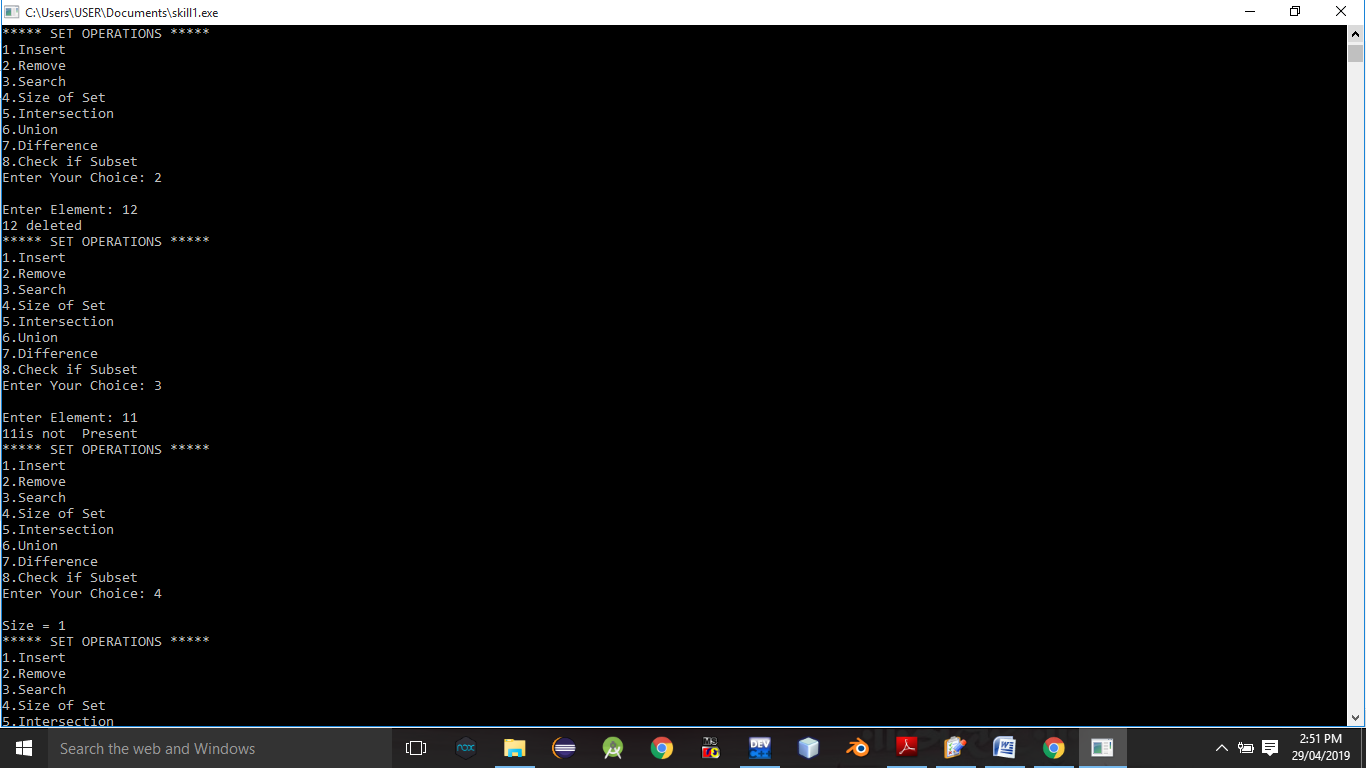
}while(choice!=0);

return 0;

}

**Output Screenshots:-**

****

****

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