**Name: Shilpa Murali**

**Student ID: 11**

Data Structure used:

1. Arrays have been used for coding this program.
2. The array with a size of 64 with indices from 0 to 63 containing the set values is used.
3. The unused array indices are filled with a default value which will not mix up with any of the values present in the set. This helps in reducing the number of iterations.

Algorithms used:

1. Adding element to a set:

A new object is constructed and the original object is copied to it. The element to be added is initialized to the corresponding array index of the new object and it is returned by value which in turn calls a copy constructor.

1. Removing an element from the set:

A new object is constructed and the calling object (\*this) is copied to it. The element to be deleted is set to the default value (-99) and new object is returned by value which in turn calls a copy constructor.

1. Union of sets(Overload with +)

A new object is constructed. The loop is iterated for 64 times and if either of the sets contains a value other than the default value of -99 the number is copied to the new object. New object is returned by value which in turn calls a copy constructor.

1. Intersection of sets(Overload with \*)

A new object is constructed. The loop is iterated for 64 times and only if both the sets contain a value other than the default value of -99 the number is copied to the new object. New object is returned by value which in turn calls a copy constructor.

1. Diffrenece of Sets:

A new object is constructed. The loop is iterated for 64 times and if both the sets contain a value other than the default value of -99 the number is set to default value(-99) in the new object. New object is returned by value which in turn calls a copy constructor.

1. Operator==:

If both the sets contain equal values in their corresponding indices then the function returns true, else it returns false.

1. Operator pre and post increment:

++i: A new object is constructed and the original object is copied to it. For every element present in the set, the element is set to the default value (-99) and the value is incremented by 1 and is inserted into the next index.

i++: A new object is constructed and the original object is copied to it. For every element present in the original set, the element is set to the default value (-99) and the value is incremented by 1 and is inserted into the next index. The new object remains unchanged and it is returned.

1. Operator pre and post decrement:

--i: A new object is constructed and the original object is copied to it. For every element present in the set, the element is set to the default value (-99) and the value is decremented by 1 and is inserted into the previous index.

i--: The function accepts an additional integer variable as a parameter to the function. : A new object is constructed and the original object is copied to it. For every element present in the original set, the element is set to the default value (-99) and the value is decremented by 1 and is inserted into the previous index. The object that was copied remains unchanged and it is returned.

1. Complement (~):

A new object is constructed and the original object is copied to it. For every element present in the set, the element is set to the default value (-99) and the indices which have a default value of -99 the value of the index is replaced in the array. The new object is copied and it is returned.

1. Conversion operator overload with():

If any of the indices of the array contains an element value other than -99, the function returns true, otherwise it returns false.

1. Conversion operator overload with(!):

If any of the indices of the array contains an element value other than -99, the function returns false, otherwise it returns true.

1. Demorgan’s law:

The +,\*,~ operators used in the above functions were used to implement the Demorgan’s law. No separate function was used to implement this.

**OUTPUT:**

a = { }

set a after adding 5 = {5,}

set a after adding 5 = {5,}

set a after adding 0 and 63 = {0,5,63,}

set b = {1,3,6,}

set b after removing 3 = {1,6,}

set b after removing 3 = {1,6,}

set b after removing 10 = {1,6,}

set b after removing 6 = {1,}

set b after removing 1 = { }

set b after adding {10,2} = {2,10,}

TESTING: iset64 operator+(const iset64& a, const iset64& b)

Set a {1,2,}

Set b {1,2,3,}

a + b = {1,2,3,}

TESTING:iset64 operator+(const iset64& a, const int b)

{1,2,}

{1,2} + 1 = {1,2,}

{1,2,}

{1,2} + 3 = {1,2,3,}

TESTING:iset64 operator+(const int b, const iset64& a)

Set a {1,2,}

1 + {1,2} = {1,2,}

Set a {1,2,}

3 + {1,2} = {1,2,3,}

TESTING:iset64& iset64::operator+=(const iset64& a)

Set b {1,2,}

Set a {1,3,}

{1,2} + {1,3} = {1,2,3,}

iset64& iset64::operator+=(const int b)

Set a {1,2,}

{1,2} + 3 = {1,2,3,}

Set a {1,2,}

Set b {3,4,}

Set c {7,8,}

Set d {1,2,3,4,5,7,8,}

TESTING: iset64 operator-(const iset64& a, const iset64& b)

Set a {1,2,}

Set b {1,2,}

a - b = { }

TESTING: iset64 operator-(const iset64& a, const iset64& b)

Set a {1,5,}

Set b {1,2,3,}

a - b = {2,3,5,}

TESTING: iset64 operator-(const iset64& a, const int b)

Set a {1,2,}

a - 3 = {1,2,}

TESTING: iset64 operator-(const int b, const iset64& a)

Set a {1,2,}

3 - a = {1,2,}

TESTING: iset64& iset64::operator-=(const iset64& a)

Set a {1,3,}

Set b {1,2,}

b -= a = {2,3,}

TESTING: iset64& iset64::operator-=(const int b)

Set a {1,2,}

a -= 3 = {1,2,}

Set a {1,2,}

Set b {2,4,}

Set c {2,8,}

Set d {1,2,4,5,8,}

TESTING: iset64 operator\*(const iset64& a, const iset64& b)

Set a {1,2,}

Set b {1,2,3,}

a \* b = {1,2,}

TESTING:iset64 operator\*(const iset64& a, const int b)

Set a {1,2,}

{1,2} \* 1 = {1,}

Set a {1,2,}

{1,2} \* 3 = { }

TESTING:iset64 operator\*(const int b, const iset64& a)

Set a {1,2,}

1 \* {1,2} = {1,}

Set a {1,2,}

3 \* {1,2} = { }

TESTING:iset64& iset64::operator\*=(const iset64& a)

Set b {1,2,}

Set a {1,3,}

{1,2} \* {1,3} = {1,}

iset64& iset64::operator\*=(const int b)

Set a {1,2,}

{1,2} \* 3 = { }

Set a {1,2,}

Set b {2,4,}

Set c {2,8,}

Set d {2,5,}

TESTING: bool operator==(const iset64& a, const iset64& b)

Set a {1,2,}

Set b {1,2,}

a == b true

{1,2,}{2,}a == b false

TESTING: bool operator!=(const iset64& a, const iset64& b)

Set a {1,2,}

Set b {1,2,}

a != b false

Set a {1,2,}

Set b {2,}

a != b true

a = {1,2,63,}

++a = {0,2,3,}

a = {1,2,63,}

acopy = {1,2,63,}

a++ = {0,2,3,}

rhs = {1,2,63,}

a = {0,2,63,}

--a = {1,62,63,}

a = {0,2,63,}

acopy = {0,2,63,}

a-- = {1,62,63,}

rhs = {0,2,63,}

a = {0,2,63,}

~a = {1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,}

ans = {1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,}

~ans = {0,2,63,}

a = {4,5,6,}

b = {5,6,8,}

aplusb = {4,5,6,8,}

aplusbbar = {0,1,2,3,7,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

abar = {0,1,2,3,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

bbar = {0,1,2,3,4,7,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

abarplusbbar = {0,1,2,3,4,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

abardotbbar = {0,1,2,3,7,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

adotb = {5,6,}

adotbbar = {0,1,2,3,4,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

Demorgan law (a+b)' = a'. b' is proved

Demorgan law (a.b)' = a' + b' is proved

a = {1,2,4,5,}

b = {2,3,5,6,}

aplusb = {1,2,3,4,5,6,}

aplusbbar = {0,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

abar = {0,3,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

bbar = {0,1,4,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

abarplusbbar = {0,1,3,4,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

abardotbbar = {0,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

adotb = {2,5,}

adotbbar = {0,1,3,4,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,}

Demorgan law (a+b)' = a'. b' is proved

Demorgan law (a.b)' = a' + b' is proved

**iset64.h**

#include "../util/util.h"

class iset64

{

private:

int\* \_arr;

void \_copy(const iset64& rhs) ;

public:

iset64();

iset64(const int\* x,int size);

~iset64();

friend iset64 operator+(const int b,const iset64& c);

friend iset64 operator+(const iset64& c,const iset64& d);

friend iset64 operator+(const iset64& d,const int c);

iset64& operator+=(const int b);

iset64& iset64::operator+=(const iset64& a);

iset64 operator-(const int b) const;

friend iset64 operator-(const iset64& a, const iset64& b);

friend iset64 operator-(const int b, const iset64& a);

iset64& iset64::operator-=(const iset64& a);

iset64& iset64::operator-=(const int b);

friend iset64 operator\*(const iset64& c,const iset64& d);

friend iset64 operator\*(const iset64& a, const int b);

friend iset64 operator\*(const int b, const iset64& a);

iset64& iset64::operator\*=(const iset64& a);

iset64& iset64::operator\*=(const int b);

iset64(const iset64& x) ;

iset64& operator=(const iset64& x) ;

friend iset64& operator++(iset64& a);

friend iset64 operator++(iset64& a,int b);

friend iset64& operator--(iset64& x);

friend iset64 operator--(iset64& a,int b);

friend iset64 operator~(const iset64& a);

friend bool operator==(const iset64& x,const iset64& y);

friend bool operator!=(const iset64& a, const iset64& b);

bool operator()(const iset64& a);

friend ostream& operator<<(ostream& o, const iset64& x) ;

operator bool() const;

};

**iset64.cpp**

#include "iset64.h"

iset64::iset64()

{

\_arr=new int [64];

for(int i=0;i<64;i++)

{

\_arr[i]=-99;

}

}

iset64::iset64(const int\* x,int size)

{

\_arr=new int [64];

for(int i=0;i<64;i++)

{

\_arr[i]=-99;

}

for(int j=0;j<size;j++)

{

\_arr[(x[j])]=x[j];

}

}

iset64::~iset64()

{

delete [] \_arr;

}

iset64 operator+(const int b,const iset64& d)

{

iset64 t(d) ;

t.\_arr[b]=b ;

return t ;

}

iset64 operator+(const iset64& d,const int c)

{

iset64 t(d);

t.\_arr[c]=c;

return t;

}

iset64 operator+(const iset64& c,const iset64& d)

{

iset64 x(c);

for(int i=0;i<64;i++)

{

if(c.\_arr[i]!=-99 || d.\_arr[i]!=-99)

{

x.\_arr[i]=i;

}

}

return x;

}

iset64 operator\*(const iset64& c,const iset64& d)

{

iset64 x;

for(int i=0;i<64;i++)

{

if(c.\_arr[i]!=-99 && d.\_arr[i]!=-99)

{

if(c.\_arr[i] == d.\_arr[i])

{

x.\_arr[i]=i;

}

}

}

return x;

}

iset64 operator\*(const iset64& a, const int b)

{

iset64 x(a);

for(int i=0;i<64;i++)

{

if(x.\_arr[i]==b)

{

x.\_arr[i]=b;

}

else

{

x.\_arr[i]=-99;

}

}

return x;

}

iset64 operator\*(const int b, const iset64& a)

{

iset64 x(a);

for(int i=0;i<64;i++)

{

if(x.\_arr[i]==b)

{

x.\_arr[i]=b;

}

else

{

x.\_arr[i]=-99;

}

}

return x;

}

iset64& iset64::operator\*=(const iset64& a)

{

for(int i=0;i<64;i++)

{

if(\_arr[i]==a.\_arr[i])

{

\_arr[i]=a.\_arr[i];

}

else

{

\_arr[i]=-99;

}

}

return \*this;

}

iset64& iset64::operator\*=(const int b)

{

for(int i=0;i<64;i++)

{

if(\_arr[i]!=b)

{

\_arr[i]=-99;

}

}

return \*this;

}

bool operator==(const iset64& a, const iset64& b)

{

int count=0;

for(int i=0;i<64;i++)

{

if((a.\_arr[i]!=-99) ||( b.\_arr[i]!=-99))

{

if(a.\_arr[i]!=b.\_arr[i])

{

return false;

}

}

}

return true;

}

iset64& operator++(iset64& x)

{

iset64 t(x);

for(int i=63;i>=0;i--)

{

if(t.\_arr[i]!=-99)

{

if(i==63)

{

x.\_arr[0]=0;

x.\_arr[i]=-99;

}

else

{

x.\_arr[i+1]=i+1;

x.\_arr[i]=-99;

}

}

}

return x;

}

iset64 operator++(iset64& a,int b)

{

iset64 t(a);

for(int i=63;i>=0;i--)

{

if(t.\_arr[i]!=-99)

{

if(i==63)

{

a.\_arr[0]=0;

a.\_arr[i]=-99;

}

else

{

a.\_arr[i+1]=i+1;

a.\_arr[i]=-99;

}

}

}

return t;

}

iset64& operator--(iset64& x)

{

iset64 t(x);

for(int i=63;i>=0;i--)

{

if(t.\_arr[i]!=-99)

{

if(i==0)

{

x.\_arr[63]=63;

x.\_arr[i]=-99;

}

else

{

x.\_arr[i-1]=i-1;

x.\_arr[i]=-99;

}

}

}

return x;

}

iset64 operator--(iset64& a,int b)

{

iset64 t(a);

for(int i=63;i>=0;i--)

{

if(t.\_arr[i]!=-99)

{

if(i==0)

{

a.\_arr[63]=63;

a.\_arr[i]=-99;

}

else

{

a.\_arr[i-1]=i-1;

a.\_arr[i]=-99;

}

}

}

return t;

}

bool operator!=(const iset64& a, const iset64& b)

{

for(int i=0;i<64;i++)

{

if(a.\_arr[i]!=-99 || b.\_arr[i]!=-99)

{

if(a.\_arr[i]!=b.\_arr[i])

{

return true;

break;

}

}

}

return false;

}

iset64& iset64::operator+=(int b)

{

\_arr[b]=b;

return \*this ;

}

iset64& iset64::operator+=(const iset64& a)

{

for(int i=0;i<64;i++)

{

if(a.\_arr[i]!=-99)

{

\_arr[i]=i;

}

}

return \*this;

}

iset64 iset64::operator-(int b) const

{

iset64 t(\*this);

t.\_arr[b]=-99;

return t;

}

iset64 operator-(const iset64& a, const iset64& b)

{

iset64 t;

for(int i=0;i<64;i++)

{

if(a.\_arr[i]!=-99 || b.\_arr[i]!=-99)

{

if(a.\_arr[i]==b.\_arr[i])

{

t.\_arr[i]=-99;

}

else

{

t.\_arr[i]=i;

}

}

}

return t;

}

iset64 operator-(const int b, const iset64& a)

{

iset64 t(a);

t.\_arr[b]=-99;

return t;

}

iset64& iset64::operator-=(const iset64& a)

{

for(int i=0;i<64;i++)

{

if(\_arr[i]!=-99 && a.\_arr[i]!=-99)

{

\_arr[i]=-99;

}

else if(a.\_arr[i]!=-99)

{

\_arr[i]=i;

}

}

return \*this;

}

iset64& iset64::operator-=(const int b)

{

for(int i=0;i<64;i++)

{

if(\_arr[i]!=-99 && b==i)

{

\_arr[i]=-99;

}

}

return \*this;

}

iset64& iset64::operator=(const iset64& s)

{

if (this != &s) {

\_copy(s) ;

}

return \*this ;

}

iset64::iset64(const iset64& s) {

\_arr=new int [64];

\_copy(s) ;

}

void iset64::\_copy(const iset64& s) {

for(int i=0;i<64;i++)

{

\_arr[i] = s.\_arr[i] ;

}

}

ostream& operator<<(ostream& o, const iset64& x)

{

bool isEmpty=true;

o<<"{";

for(int i=0;i<64;i++){

if(x.\_arr[i]!=-99)

{

isEmpty=false;

o << x.\_arr[i] << ",";

}

}

if(isEmpty==true)

{

o<<" ";

}

o<<"}";

return o ;

}

iset64 operator~(const iset64& a){

iset64 t(a);

for(int i=0;i<64;i++){

if(a.\_arr[i]!=-99){

t.\_arr[i]=-99;

}

else if(a.\_arr[i]==-99){

t.\_arr[i]=i;

}

}

return t;

}

iset64::operator bool() const

{

bool set=false;

for(int i=0;i<64;i++)

{

if(\_arr[i]!=-99)

{

set=true;

break;

}

}

return set;

}