/\*

1. Implement a CPP program based on SET operations using OOP concepts. (UNION, INTERSECTION, subtraction)

\*/

#include<bits/stdc++.h>

using namespace std;

class SET{

    /\*

        STEPS:

            1. Default constructor which will initialise the object with a NULL SET. DONE

            2. Parameterized constructor which will initialise the object with a given array. DONE

            3. Input function which will take the input of that SET from the user. DONE

            4. INTERSECTION function which will return an object of the same class. DONE

            5. UNION function which will return an object of the same class. DONE

            6. Difference between two SETs require an overloaded operator function (operator-) DONE

    \*/

private:

    int \*a;

    int n;

public:

    SET(){

        // Initialize the array with a null SET.

        a = NULL;

    }

    void makeset(){

        // Remove duplicates

        int \*\*temp;

        temp = new int\*[n];

        for(int i=0; i<n; i++)

            temp[i] = new int(a[i]);

        // Check for duplicates

        int null\_count=0;

        for(int i=0; i<n; i++){

            for(int j=i+1; j<n; j++){

                if(temp[i] != NULL && \*temp[i] == \*temp[j]){

                    temp[j] = NULL;// Remove that element

                    null\_count++;

                }

            }

        }

        a = new int[n - null\_count];

        int j=0;

        for(int i=0; i<n; i++)

            if(temp[i] != NULL){

                a[j++] = \*temp[i];

            }

        n-= null\_count;

    }

    SET(int \*elements, int len){

        // Initialize the array of this SET with the elements given.

        this->n = len;

        this->a = new int [n];

        for(int i=0; i < this->n; i++){

            this->a[i] = elements[i];

        }

        // After creating a set from the given array, we must check for duplicate elements and remove if found

        this->makeset();

    }

    void input(int sr){ // sr is serial number of set

        // Take the no of elements first then take those many inputs.

        cout<<"\nEnter the number of elements for SET no "<<sr<<":\n";

        cin>>n;

        a = new int [n];

        cout<<"Enter the SET:\n";

        for(int i=0; i < n; i++){

            cout<<"Enter element no "<<i+1<<":\n";

            cin>>a[i];

        }

        makeset();

    }

    void display(){

        cout<<"\n";

        for(int i=0; i < n; i++)

            cout<<this->a[i]<<"\t";

    }

    SET Interesct(SET& x){

        // There are two arrays: One from which this function is being called (this->a), and the other is "x".

        // Take the common elements from these two arrays, create an object of this array and return that object.

        int len\_self = n;

        int len\_other = x.n;

        vector<int> v;

        for(int i=0; i < len\_self; i++)

            for(int j=0; j < len\_other; j++)

                if(this->a[i] == x.a[j]){

                    if(count(v.begin(), v.end(), this->a[i]))

                        continue;

                    else{

                        //cout<<this->a[i]<<endl;

                        v.push\_back(this->a[i]);

                    }

                }

        // Create an object of this common array

        SET c(v.data(), v.size());

        return c;

    }

    SET Union(SET& x){

        // There are two arrays: One from which this function is being called (this->a), and the other is "x".

        // Combine two arrays and apply makeset() on the new array.

        // Create a new set of size this->n + x.n

        int \*temp = new int[this->n + x.n];

        int index = 0;  // Index for this temp set

        // Copy the elements of set this->a then set x.a

        for(int i=0; i < this->n; i++)

            temp[index++] = this->a[i];

        for(int i=0; i < x.n; i++)

            temp[index++] = x.a[i];

        // Now create a new Set object from this array which will then call makeset() on this array.

        SET s(temp, this->n + x.n);

        return s;

    }

    SET operator-(SET &x){

        // There are two arrays: One from which this function is being called (this->a), and the other is "x".

        // The calling statement was (this->SET - X) OR (this->a - x.a)

        // Take the common elements from both arrays

        // Remove all these common elements from this->a

        // Take common elements

        SET common = Interesct(x);

        // Common elements are in common->a;

        // Iterate on this->a and common.a and remove any element from "this->a" if found in common.a

        int \*\*temp;

        temp = new int\*[this->n];

        for(int i=0; i < this->n; i++)

            temp[i] = new int(this->a[i]);  // Creating a array mapping

        // Iterate on temp

        int common\_count = 0;

        for(int i=0; i < this->n; i++){ // For each element, check if it is present in x.a

            for(int j=0; j < x.n; j++)

                if(temp[i] != NULL && x.a[j] == \*temp[i]){  // Set this pointer to null as we do not need this element

                    temp[i] = NULL;

                    common\_count++;

                }

        }// Common elements are removed.

        // New array is of length this->n - common\_count

        int \*final\_array = new int[this->n - common\_count];

        int index = 0;

        for(int i=0; i < this->n; i++)

            if(temp[i] != NULL)

                final\_array[index++] = \*temp[i];

        // New array is created. Create a set from this array

        SET difference(final\_array, index);

        return difference;

    }

};

int main(){

    SET s1, s2;

    s1.input(1);

    cout<<"SET No 1 is:\n";

    s1.display();

    s2.input(2);

    cout<<"SET No 2 is:\n";

    s2.display();

    SET s3 = s1.Interesct(s2);

    cout<<"\nIntersection of these sets is:\n";

    s3.display();

    s3 = s1.Union(s2);

    cout<<"\nUnion of these sets is:\n";

    s3.display();

    s3 = s1 - s2;

    cout<<"\nS1 - S2 is:\n";

    s3.display();

}

