#include<bits/stdc++.h>

using namespace std;

class Polynomial{

    public:

    int\* poly\_no;

    int capacity;

    //Default Constructor

    Polynomial(){

        this->poly\_no = new int[6];

        this->capacity = 5;

    }

    //Paramaterized constructor

    Polynomial(int capacity){

        this->poly\_no = new int[capacity+1];

        this->capacity = capacity;

    }

    //copy constructor

    Polynomial(Polynomial const &P){

        int \*P\_res = new int[P.capacity+1];

        for(int i=0;i<P.capacity;i++)

            P\_res[i]=P.poly\_no[i];

        this->poly\_no = P\_res;

        this->capacity = P.capacity;

    }

    //set coefficient function

    void setCoefficient(int deg, int coef){

        if(deg>capacity){

            int new\_cap = deg;

            int \*newdeg = new int[new\_cap+1];

            //copying from original to new

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

                newdeg[i]=poly\_no[i];

            }

            this->poly\_no = newdeg;

            this->capacity = new\_cap;

            poly\_no[deg]=coef;

        }else{

            poly\_no[deg] = coef;

        }

    }

    //overload + operator

    Polynomial operator +(Polynomial const &P){

        int max\_cap = max(capacity,P.capacity);

        Polynomial P\_res(max\_cap);

        for(int i=0;i<=max\_cap;i++){

            if(i<=capacity && i<=P.capacity)

                P\_res.poly\_no[i]=this->poly\_no[i]+P.poly\_no[i];

            else if(i<=capacity)

                P\_res.poly\_no[i]=this->poly\_no[i];

            else

                P\_res.poly\_no[i]=P.poly\_no[i];

        }

        return P\_res;

    }

    //overload - operator

    Polynomial operator -(Polynomial const &P){

        int max\_cap = max(capacity,P.capacity);

        Polynomial P\_res(max\_cap);

        for(int i=0;i<=max\_cap;i++){

            if(i<=capacity && i<=P.capacity)

                P\_res.poly\_no[i]=this->poly\_no[i]-P.poly\_no[i];

            else if(i<=capacity)

                P\_res.poly\_no[i]=this->poly\_no[i];

            else

                P\_res.poly\_no[i]=P.poly\_no[i];

        }

        return P\_res;

    }

    //overload \* operator

    Polynomial operator \*(Polynomial const &P){

        int max\_cap = this->capacity+P.capacity;

        Polynomial P\_res(max\_cap);

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

            for(int j=0;j<=P.capacity;j++){

                P\_res.poly\_no[i+j]+=poly\_no[i]\*P.poly\_no[j];

            }

        }

        return P\_res;

    }

    //overload = operator

    void operator =(Polynomial const &P){

        int \*P\_res = new int[P.capacity+1];

        for(int i=0;i<P.capacity;i++)

            P\_res[i]=P.poly\_no[i];

        this->poly\_no = P\_res;

        this->capacity = P.capacity;

    }

    //print function

    void print(){

        for(int i=0;i<capacity;i++){

            if(poly\_no[i]!=0)

                cout<<poly\_no[i]<<"x"<<i<<" ";

        }

        cout<<endl;

    }

};

int main(){

    int N,M,C;

    cin>>N;

    int \*degree\_1 = new int[N];

    int \*coeff\_1 = new int[N];

    for(int i=0;i<N;i++) cin>>degree\_1[i];

    for(int i=0;i<N;i++) cin>>coeff\_1[i];

    Polynomial poly\_1;

    for(int i=0;i<N;i++)

        poly\_1.setCoefficient(degree\_1[i],coeff\_1[i]);

    cin>>M;

    int \*degree\_2= new int[M];

    int \*coeff\_2 = new int[M];

    for(int i=0;i<M;i++) cin>>degree\_2[i];

    for(int i=0;i<M;i++) cin>>coeff\_2[i];

    Polynomial poly\_2;

    for(int i=0;i<M;i++)

        poly\_2.setCoefficient(degree\_2[i],coeff\_2[i]);

    cin>>C;

    switch(C){

        //sum

        case 1:

        {

            Polynomial res = poly\_1+poly\_2;

            res.print();

            break;

        }

        //subtract

        case 2:

        {

            Polynomial res = poly\_1+poly\_2;

            res.print();

            break;

        }

        //multiply

        case 3:

        {

            Polynomial res = poly\_1\*poly\_2;

            res.print();

            break;

        }

        //assignment operator

        case 4:

        {

            Polynomial poly\_3(poly\_1);

            if(poly\_3.poly\_no == poly\_1.poly\_no){

                cout<<"false"<<endl;

            }

            else{

                cout<<"true"<<endl;

            }

            break;

        }

        //copy constructor

        case 5:

        {

            Polynomial poly\_4(poly\_1);

            if(poly\_4.poly\_no == poly\_1.poly\_no){

                cout<<"false"<<endl;

            }

            else{

                cout<<"true"<<endl;

            }

            break;

        }

    }

    return 0;

}