//#include <iostream>

//using namespace std;

//bool somePredicate(double x)

//{

// return x > 0;

//}

// Return true if the somePredicate function returns false for at

// least one of the array elements; return false otherwise.

**bool** anyFalse(**const** **double** a[], **int** n)

{

**if** (n == 0)

**return** **false**;

**if** (somePredicate(\*a))

**return** anyFalse(a+1, n-1);

**else**

**return** **true**;

**return** **false**;

}

// Return the number of elements in the array for which the

// somePredicate function returns true.

**int** countTrue(**const** **double** a[], **int** n)

{

**if** (n == 0)

**return** 0;

**if** (somePredicate(\*a))

**return** 1 + countTrue(a+1, n-1);

**else**

**return** 0 + countTrue(a+1, n-1);

}

// Return the subscript of the first element in the array for which

// the somePredicate function returns true. If there is no such

// element, return -1.

**int** firstTrue(**const** **double** a[], **int** n)

{

**if** (n == 0)

**return** -1;

**if** (somePredicate(\*a))

**return** 0;

**if** (firstTrue(a+1, n-1) == -1)

**return** -1;

**else**

**return** firstTrue(a+1, n-1) + 1;

}

// Return the subscript of the smallest element in the array (i.e.,

// return the smallest subscript m such that a[m] <= a[k] for all

// k from 0 to n-1). If the function is told to examine no

// elements, return -1.

**int** positionOfSmallest(**const** **double** a[], **int** n)

{

**if** (n < 1)

**return** -1;

**if** (n == 1)

**return** 0;

n--;

**if** (a[n] < a[positionOfSmallest(a, n)])

**return** n;

**else**

**return** positionOfSmallest(a, n);

}

// If all n2 elements of a2 appear in the n1 element array a1, in

// the same order (though not necessarily consecutively), then

// return true; otherwise (i.e., if the array a1 does not contain

// a2 as a not-necessarily-contiguous subsequence), return false.

// (Of course, if a2 is empty (i.e., n2 is 0), return true.)

// For example, if a1 is the 7 element array

// 10 50 40 20 50 40 30

// then the function should return true if a2 is

// 50 20 30

// or

// 50 40 40

// and it should return false if a2 is

// 50 30 20

// or

// 10 20 20

**bool** contains(**const** **double** a1[], **int** n1, **const** **double** a2[], **int** n2)

{

**if** (n2 == 0)

**return** **true**;

**if** (n1 == 0)

**return** **false**;

**if** (\*a2 == \*a1)

**return** contains(a1+1, n1-1, a2+1, n2-1);

**else**

**return** contains(a1+1, n1-1, a2, n2);

}

//int main()

//{

// double array[7] = {10, 50, 40, 20, 50, 40, 30};

// double array2[3] = {10, 20, 20};

// if (anyFalse(array, 5))

// cout << "at least one negative";

// else

// cout << "all positive";

// cout << endl << countTrue(array, 7);

// cout << endl << firstTrue(array, 7);

// cout << endl << positionOfSmallest(array, 7);

// if (contains(array, 7, array2, 3))

// cout << endl << "contains";

// else

// cout << endl << "doesnt contain";

//}

// If all n2 elements of a2 appear in the n1 element array a1, in

// the same order (though not necessarily consecutively), then

// return true; otherwise (i.e., if the array a1 does not contain

// a2 as a not-necessarily-contiguous subsequence), return false.

// (Of course, if a2 is empty (i.e., n2 is 0), return true.)

// For example, if a1 is the 7 element array

// 10 50 40 20 50 40 30

// then the function should return true if a2 is

// 50 20 30

// or

// 50 40 40

// and it should return false if a2 is

// 50 30 20

// or

// 10 20 20

**bool** contains(**const** **double** a1[], **int** n1, **const** **double** a2[], **int** n2)

{

**if** (n2 == 0)

**return** **true**;

**if** (n1 == 0)

**return** **false**;

**if** (\*a2 == \*a1)

**return** contains(a1+1, n1-1, a2+1, n2-1);

**else**

**return** contains(a1+1, n1-1, a2, n2);

}