# Project 4: Need Covering

## Problem Description:

Relief organizations, after a tragedy, generate an array of needed items, where each item is represented by a number. (e.g., 5 might represent a pallet of bagged rice.)

Similarly, supply depots maintain their inventory in similar fashion: an array of items currently in stock, where each item is represented by a number. Everyone uses the same numbering system. So, if a supply depot's inventory array contains two 5's, that means it has two pallets of bagged rice in stock.

Given two integer arrays, A (aka Need) and B (aka Inventory), array B is said to cover (or satisfy) Array A if for every item in array A, there exists a unique item in array B that can be matched (has equal value) to the item in A. A single item in B cannot be matched to more than one item in A.

Note: **Both arrays may contain duplicate entries.**

It is important to know when a depot's inventory covers a relief organization's need list - one stop shopping in the provision of much needed relief supplies.

Your program is to read in two integer arrays (Need and Inventory) and determine if the supply depot represented by array Inventory (B) covers (satisfies or fulfills) the needs as represented by array Need (A). Or put another way: does array B cover array A?

For a B-level project, you can assume both arrays will only contain positive integer values (see levels of achievement for the requirements for an A-level project).

**Do not assume the inputs for either array are sorted.**

If Inventory array (B) covers the Needs array (A), your program will output

**satisfies**

otherwise it should output

**does not satisfy**

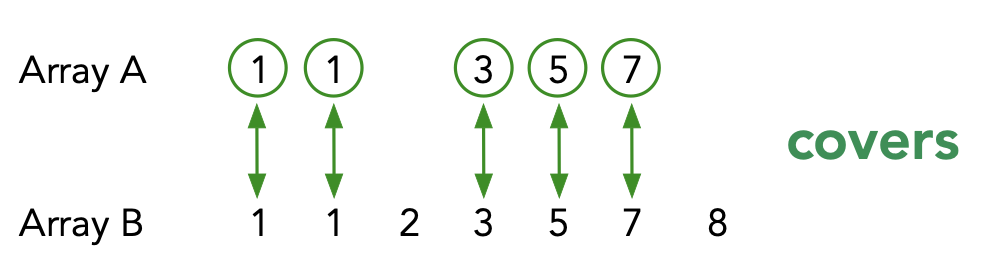
Your program should not print anything else to the screen. Print one of the two messages above spelled/formatted exactly that way. You will lose points for not following these print instructions.

For example, if we have;

array A: 1, 1, 3, 5, 7

array B: 1, 1, 2, 3, 4, 5, 7, 8

then array B does cover array A because we can find a single element in B that matches each element in A.

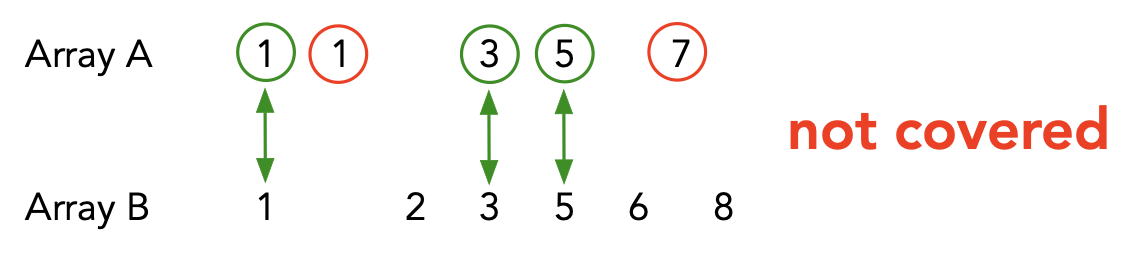


However, in the example

            array A:           1, 1, 3, 5, 7

            array B:            1, 2, 3, 5, 6, 8

array B does not cover A because array A has two 1's and there is only one 1 in array B.  There is a "1" in array A that does not have a match in B.  Also array A has a "7" while array B does not.



## Input Data Format:

The input will be read from the keyboard -- we call this reading from stdin (reading from standard in). However, we will leverage the power of the linux command line to redirect input from files into stdin so that the program thinks it is reading from the keyboard but instead the data is coming from a file(s).

The input will come in the form of:

*numA*

*A[0]*

*A[1]*

*...*

*A[numA-1]*

*numB*

*B[0]*

*B[1]*

*...*

*B[numB-1]*

where the number of elements of A (numA) is given first, followed by the data for array A, one per line.  Then immediately the number of elements of B (numB) is given next followed by the data for B, again one per line. Your program should read in numA, then create an integer array of the right size for A, then use a loop to read in the next numA integers into the array.  Then it does the same for array B (first read numB, then read numB integers into the array). numA and numB will always be between 1 and 10,000 inclusive. The values in the arrays will be any legal positive int value in C++.

See the provided sample data files: test1a.txt and test1b.txt; test2a.txt and test2b.txt

While your program reads from stdin (using cin) we don't want to have to type in all the data each time we run the program, so we will use linux pipes to solve the problem.  We will put the data for array A in one file (preceded by numA) and the data for array B in a second file (preceded by numB). Again, see the two sample files for an example.

The **cat** command "concatenates" data in text files and prints it to the screen. Try the following command:

**cat test1a.txt test1b.txt**

What do you see?   You should see the contents of A and then immediately the contents of B printed to the screen.

Now we will "pipe" these contents to the input of our C++ program.

**g++ -o cover.exe cover.cpp**

**cat test1a.txt test1b.txt | ./cover.exe**

This should cause one's compiled program (named cover) to receive its input from the **cat** command instead of from the keyboard.  We are using linux pipes (|) to change the keyboard input to the file input instead, while our program still thinks it is reading from stdin!

## Requirements:

You are to write a boolean function (returns true or false) called **coversNeeds** that takes the Needs array (A), the size of Needs array, the Inventory array(B), and the size of the Inventory array as its four parameters.  It is to return **true**or **false** depending on whether or not the given inventory list (array B) covers the needed list of supplies (array A).

The main() function can do the job of reading the input, calling the boolean function, and displaying the output.

You are not limited to these two functions. By applying the principles of functional decomposition (top-down design) you will probably uncover the need for additional helper functions.

## Levels of Achievement:

* D range: A non-working (but still well documented) program. Points are awarded based on progress made.
* C range: A well documented, working program, but not well designed/decomposed into small individual functions.
* B range: A well documented, well designed, working program that works for arrays containing positive integers.
* A range: Relax the restriction that the input arrays only contain positive integers and can instead contain any integer. To solve this version you will need to sort both arrays and use two position markers to process your way through the two arrays. Implement your own sort function (e.g. Selection Sort or Insertion Sort).

## Learning Goals:

* Practice with C++ Arrays
* Complex Algorithm Development
* Practice with both Iterative and Functional Decomposition
* Practice Creating Test Suites