# 01. Warm Winter

**You can test your solutions in** [**Judge**](https://judge.softuni.org/Contests/Practice/Index/2824#0)**.**

*Anni started her small business. She made an Etsy shop named “Warm Winter”, where she sells her matching sets of winter hats and scarfs. Help her make the sets and figure out the prices of the sets in store.*

First, you will be given **a sequence of integers representing the hats**. Afterward, you will be given another **sequence of integers representing the scarfs**.

Check all of the hats and scarves to make sets. Take **the last given hat**, and the **first given scarf** and check **if the hat is bigger than the scarf** and if it is – you have to **create a set. A set is created when you add the value of the hat to the value of the scarf (that is the price of the set)**. If you have a set, **remove both** the hat and the scarf from their collections.

**If the scarf’s value is bigger** – **remove the hat and check the next one**.

If their values **are equal** – **remove the scarf** and **increment** the value of the hat with **1**

Mary wants to give her mother **the most expensive set size**, so you have to find out which one it is and print it in the following format: "**The most expensive set is: {maxPriceSet}**"

Afterward **print the created sets** from **the first added to the last,** separated by a space.

### Input

* On the **first line** of input, you will receive the integers, representing the **hats,** **separated** by a **single space**.
* On the **second line** of input, you will receive the integers, representing the **scarfs**, **separated** by a **single space**.

### Output

* On the first line of output - print the biggest set in the format specified above.
* On the second line - print the sets, separated by a single space **in the order specified above.**

### Constraints

* All of the given numbers will be valid integers in the range [1, 10000].
* There will always be at least 1 set.
* Allowed time/memory: 100ms/16MB.

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **10 8 7 13 8 4**  **4 7 3 6 4 12** | **The most expensive set is: 16**  **15 16 13 12** | First, we take the last given hat – 4 and the first given scarf – 4. They are equal, so we have to remove the scarf and increment the hat with 1. The hat becomes 5 and the collection looks like this  Hats: 10 8 7 13 8 5  Scarfs: 7 3 6 4 12  Next, we take the hat with value 5 and the scarf with value 7 – the scarf is bigger, so we **remove** the **hat** and the collections should look like this:  Hats: 10 8 7 13 8  Scarfs: 7 3 6 4 12  After that, we the hat 8 and the scarf 7 – the hat is bigger, so we have our first **set** with a value of 15. In the end, we have to print the most expensive set, which in this case is with value 16, and the collection of sets, that we have created. |
| **9 5 4 7 8 5 2 6 9**  **1 4 5 7 9 6 3 5 4 7** | **The most expensive set is: 16**  **10 10 15 16** |  |

# 02. Warships

**You can test your solutions in** [**Judge**](https://judge.softuni.org/Contests/Practice/Index/2813#1)**.**

*You have been tasked with the development of an online version of the famous Warships game. You start by developing a console version.*

We get as input **the size** of the **field** in which our Warships are situated. The field is **always a square**. After that, we will receive the coordinates for the attacks for each player in pairs of integers (row, column). If there is an attack on invalid coordinates **- ignore the attack**. The possible characters that may appear on the screen are:

* **\*** – a regular position on the field
* **<** – ship of the first player.
* **>** – ship of the second player
* **#** –a sea mine that explodes when attacked

Each time when an attack hits a ship or a mine **replace it with 'X'**. Keep track of the **count of ships remaining for each player**. If a player hits a mine it explodes destroying all ships in adjacent fields (friend or foe). If a player destroys all enemy ships, the program stops and you have to print the following message: **"Player {One/Two} has won the game! {totalCountShipsDestroyed} ships have been sunk in the battle."**

If the list of attack commands endsbefore any of the players has won, you have to print the following message:

**"It's a draw! Player One has {countOfShips} left. Player Two has {countOfShips} left."**

## Input

* **Field size** – an integer number.
* **Attack commands** – All attack coordinates will be received on a single line. There will be a comma (,) between each set of coordinates and whitespace (" ") between every two integers representing the row and column to attack.
* **The field: some of the following characters (\*, <, >, #),** separated by whitespace (" ");

## Output

* There are three types of output:
  + If Player One wins, print the following output: **"Player One has won the game! {totalCountShipsDestroyed} ships have been sunk in the battle."**
  + If player Two wins, print the following output: **"Player Two has won the game! {totalCountShipsDestroyed} ships have been sunk in the battle."**
  + If there are no more commands and none of the above cases had happened, you have to print the following message: **"It's a draw! Player One has {countOfShips} ships left. Player Two has {countOfShips} ships left."**

## Constraints

* The **field size** will be a 32-bit integer in the range [4 … 2 147 483 647].
* Player One always starts first.
* A player will **never attack** the coordinates of one of **his ships**.
* There will be **no incomplete sets** of attack coordinates – e.g. – **"**0 1,2 3,2**".**
* There will never be 2 mines in adjacent fields.
* There will not be a test where both players lose their last ships in the same turn.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 5  0 0,-1 -1,2 2,4 4,4 2,3 3,3 6  # < \* < \*  > > \* < \*  \* \* > \* \*  < \* \* \* \*  \* \* > \* \* | Player One has won the game! 5 ships have been sunk in the battle. | Player One attacks first at coordinates 0,0 and hits a mine. All adjacent fields (including diagonally) are blasted. The enemy ships at 1,0 and 1,1 are destroyed as well as the friendly ship at 0,1. The field now looks like this:  # X \* < \*  X X \* < \*  \* \* > \* \*  < \* \* \* \*  \* \* > \* \*  Player Two attacks on invalid coordinates so we ignore it.  Player One attacks 2,2 and destroys an enemy ship.  Player Two attacks 4,4 and misses.  Player One attacks 4,2 and destroys the last enemy ship. The game is over so we ignore any remaining attack coordinates and print messages. The final state of the map is the following:  X X \* < \*  X X \* < \*  \* \* X \* \*  < \* \* \* \*  \* \* X \* \* |
| 6  0 0,1 0,5 5  \* \* \* \* \* \*  < \* \* \* < \*  \* \* > > \* \*  \* \* \* \* \* \*  \* \* \* \* \* \*  \* \* \* \* \* \* | It’s a draw! Player One has 1 ships left. Player Two has 2 ships left. |  |
| 5  1 1,0 3,2 1,1 3,4 1,3 0  # \* \* < \*  \* \* \* < \*  \* \* > \* \*  < \* \* > \*  \* \* > \* \* | Player Two has won the game! 3 ships have been sunk in the battle. |  |

# 03. VetClinic

**You can test your solutions in** [**Judge**](https://judge.softuni.org/Contests/Practice/Index/2498#2)**.**

## Preparation

Download the skeleton provided in Judge. **Do not** change the **StartUp** class or its **namespace**.

## Problem description

Your task is to create a repository, which stores items by creating the classes described below.

First, write a C# class **Pet** with the following properties:

* **Name: string**
* **Age: int**
* **Owner: string**

The class **constructor** should receive **name, age** and **owner.** The class should override the **ToString()** method in the following format:

**"Name: {Name} Age: {Age} Owner: {Owner}"**

**Next**, write a C# class **Clinic** that has **data** (a collection, which stores the Pets). All entities inside the repository have the **same properties**. Also, the **Clinic** class should have those properties:

* **Capacity: int**

The class **constructor** should receive **capacity**, also it should initialize the **data** with a new instance of the collection**.** Implement the following features:

* Field **data** – **collection** that holds added pets
* Method Add(Pet pet) – **adds** an **entity** to the data **if** **there** **is** an **empty cell** for the pet.
* Method Remove(string name) – removes the pet by **given name,** if such **exists**, and **returns bool**.
* Method **GetPet(string name, string owner)** – returns the pet with the **given name** and **owner** or **null if no such pet exists**.
* Method GetOldestPet() – returns the oldest Pet.
* Getter Count – **returns** the **number** of pets.
* **GetStatistics()** – **returns** a **string** in the following **format**:
  + **"The clinic has the following patients:  
    Pet {Name} with owner: {Owner}  
    Pet {Name} with owner: {Owner}**

**(…)**"

## Constraints

* The **combinations** of **names** and **owners** will **always be unique**.
* The **age** of the pets will always be **positive**.

## Examples

This is an example of how the **Clinic** class is **intended to be used**.

|  |
| --- |
| Sample code usage |
| // Initialize the repository  Clinic clinic = new Clinic(20);  // Initialize entity  Pet dog = new Pet("Ellias", 5, "Tim");  // Print Pet  Console.WriteLine(dog); // Ellias 5 (Tim)  // Add Pet  clinic.Add(dog);  // Remove Pet  Console.WriteLine(clinic.Remove("Ellias")); // True  Console.WriteLine(clinic.Remove("Pufa")); // False  Pet cat = new Pet("Bella", 2, "Mia");  Pet bunny = new Pet("Zak", 4, "Jon");  clinic.Add(cat);  clinic.Add(bunny);  // Get Oldest Pet  Pet oldestPet = clinic.GetOldestPet();  Console.WriteLine(oldestPet); // Zak 4 (Jon)  // Get Pet  Pet pet = clinic.GetPet("Bella", "Mia");  Console.WriteLine(pet); // Bella 2 (Mia)  // Count  Console.WriteLine(clinic.Count); // 2  // Get Statistics  Console.WriteLine(clinic.GetStatistics());  //The clinic has the following patients:  //Bella Mia  //Zak Jon |

## Submission

Zip all the files in the project folder except **bin** and **obj** folders