# Meeting



The first line will give you **a sequence of integers representing males**. Afterward, you will be given another **sequence of integers representing females**.

You have to start from the **first female** and try to match it with the **last male**.

* If their **values** are **equal**, you have to **match them** and **remove both** of them. Otherwise, you should **remove only the female** and **decrease** the **value** of the **male** by **2**.
* If someone’s value is **equal to or below** **0**, you should **remove him/her** from the records **before** trying to **match** him/her with anybody.
* Special case - if someone’s **value is divisible by 25** **without remainder**, you should **remove** **him/her** **and** the **next person** of the **same gender**.

You need to **stop** **matching** people when you have **no more females or males**.

### Input / Constraints

* On the **first line**, you will receive the integers, representing the **males**, **separated** by a **single space**.
* On the **second line,** you will receive the integers, representing the **females**, **separated** by a **single space**.
* All of the given numbers will be valid integers in the range **[-100, 100]**.

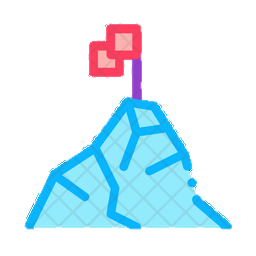
### Output

* On the first line - print the number of successful matches:
  + "**Matches: {matchesCount}"**
* On the second line - print all males left:
  + If there are no males: "**Males left: none**"
  + If there are males: "**Males left: {male1}, {male2}, {male3}, (…)**"
* On the third line - print all females left:
  + If there are no females: "**Females left: none**"
  + If there are females: "**Females left: {female1}, {female2}, {female3}, (…)**"

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **3 6 9 12**  **12 9 6 1 25 25** | **Matches: 3**  **Males left: 1**  **Females left: none** | The first pair is the **first female** with a value of 12 and the **last male** of value 12, their **values are equal**, so we **match them,** therefore - **remove them** from the **records**. Then we have **two more matches** (9 == 9 and 6 == 6). But the value of the **next male is 3** and the value of the **next female is 1**, it’s **not a match** and we **remove** the **female** and **reduce** the **male’s value** by 2. We have a **female** whose **value** is **25** and we have to **remove** **her** and the **next female**. Then, we **print** the desired **output**. |
| **3 0 3 6 9 0 12**  **12 9 6 1 2 3 15 13 4** | **Matches: 4**  **Males left: none**  **Females left: 15, 13, 4** |  |

# Throne Conquering



*Paris has entered Sparta and he has to fight to abduct the wife of Menelaus, Helen.*

After Paris got into Sparta, he has to fight his way to Helen’s chamber. To do that, he has to walk through the city where dangerous enemies are watching out for threats, but he also has to be careful not to get exhausted and not be able to proceed with his mission. If Paris successfully reaches her chamber, they safely escape from the Spartans.

A standard field of Sparta looks like this:

|  |  |
| --- | --- |
| **Field of Sparta** | **Legend** |
| ------H--- -------S-- --S------- ---------- -----P---- | P 🡺 **Paris**, the player character  S🡺 **Spartan, enemy**  H 🡺 **Helen**  - 🡺 **Empty space** |

Each turn proceeds as follows:

* **First**, Spartan **spawns** on the given indices.
* **Then, Paris** moves in a direction, which **decreases** his energy by 1.
  + It can be "**up**", "**down**", "**left**", "**right**".
  + If Paris tries to move **outside** of the field, he **doesn’t** move but **still** has his energy **decreased**.
* If an enemy is in the **same cell** where Paris moves, Paris fights him, which **decreases** his energy by 2. If Paris’ energy **drops** at 0 or below, he **dies** and you should mark his position with ‘**X**’.
* If Paris **kills** the enemy successfully, the enemy **disappears**.
* If Paris reaches the **index** where **Helen** is, **they both run away** (disappear from the field)**, even if his energy is 0 or below.**

## Input

* On the **first line** of input, you will receive **e** – **the energy** Paris has.
* On the **second line** of input, you will receive **n** – the **number of rows** the field of Sparta will consist of  
  range: **[5-20].**
* On the next **n lines**, you will receive how each row looks.
* Then, **until** Paris dies, or reaches Helen, you will receive a **move command** and **spawn row and column**.

## Output

* If Paris is **running out of energy**, print "Paris died at {row};{col}."
* If Helen is **abducted**, print "Paris has successfully abducted Helen! Energy left: {energy}"
* Then, in all cases, **print** the **final state of the field** on the **console**.

## Constraints

* The field will always be **rectangular**.
* Paris will **always** run out of energy or **reach Helen**.
* There will be **no case** with spawn on **invalid** indices.
* There will be **no case** with **two enemies on the same cell**.
* There will be **no case** with enemy **spawning** on the indices **where Paris is**.
* There will be **no case** with enemy **spawning** on the indices **where Helen is**.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 100  5  --H--  -----  -----  -----  --P--  up 3 0  up 3 1  up 3 2  up 3 3 | Paris has successfully abducted Helen! Energy left: 96  -----  -----  -----  SSSS-  ----- | Turn 1: An enemy spawns at [3;0], Paris moves to [3;2], and his energy decreases by 1.  Turn 2: An enemy spawns at [3;1], Paris moves to [2;2], and his energy decreases by 1.  Turn 3: An enemy spawns at [3;2], Paris moves to [1;2], and his energy decreases by 1.  Turn 4: An enemy spawns at [3;3], Paris moves to [0;2], his energy decreases by 1, but he also moves to the index where Helen is – they both run away. |
| 3  5  --H--  -----  -----  -----  --P--  up 3 2 | Paris died at 3;2.  --H--  -----  -----  --X--  ----- | Turn 1: An enemy spawns at [3;2], Paris moves to [3;2], his energy decreases by 1, and fights the enemy at that index. Paris’ energy is decreased by 2, dropping it to 0 or below => Paris dies. |
| 3  5  --H--  -----  -----  -----  --P--  left 1 0  down 2 0  up 3 0 | Paris died at 3;1.  --H--  S----  S----  SX---  ----- | Turn 1: An enemy spawns at [1;0], Paris moves to [4;1], and his energy decreases by 1.  Turn 2: An enemy spawns at [2;0], Paris tries to move down, but [5;1] is an invalid index, so he stays at [4;1]. His energy still decreases.  Turn 3: An enemy spawns at [3;0], Paris moves to [3;1], his energy drops to 0 and he cannot continue his mission. |

# Water Adventure



*You are told that there is an aquarium near your home, which is an attraction that should not be missed, so you decide to visit it. To remember the adventure, you decide to make a report for the aquarium.*

## Preparation

Download the skeleton provided in Judge. Do not change the packages.

**Pay attention to the name of the package, all the classes, their fields, and methods the same way they are presented in the following document. The fields you create must be the private access modifier. It is also important to keep the project structure as described above.**

## Problem description

Your task is to create a repository (aquarium) that stores departments by creating the classes, described below.

### Fish

First, write a Java **class**, called **Fish** with fields:

* **name: String**
* **color: String**
* **fins: int**

The **constructor** of the Fish class should receive a **name, color,** and **fins**.

The class should also have the following methods:

* Getter **getName()**
* Getter **getColor()**
* Getter **getFins()**
* Override **toString()** method in the format:

**"Fish: {name}**  
**Color: {color}**  
**Number of fins: {fins}"**

### Aquarium

The next step is to write an **Aquarium** class that has a **collection** of an object of type **Fish** with the corresponding **unique** **name** of a fish. The name of the collection should have the name **fishInPool**. All the entities of the **fishInPool** collection have the **same** fields. The Pool has also some additional fields:

* **name: String**
* **capacity: int**
* **size: int - the volume of the pool**

The **constructor** of the Aquarium class should receive the **name, capacity,** and **size**, also you should initialize the **collection** of fish with a new instance.

Implement the following features:

* Getter **getName()**.
* Getter **getCapacity()**.
* Getter **getSize()**.
* Getter **getFishInPool()** -returns the **number** of fish in the current pool.
* Method **add(Fish fish)** - add the entity **if** there **isn't** a fish with the same **name** and **if** there is **enough** **space** for it.
* Method **remove(String name)** - removes a fish from the pool with the given **name**, if such **exists,** and returns a **boolean** if the deletion is successful.
* Method **findFish(String name)** - returns a **fish** with the given name, **it** doesn't exist return **null**.
* Method **report()** - returns information about the aquarium and the fish inside it in the following format:

**"Aquarium: {name} ^ Size: {size}**

**{Fish1}**

**{Fish2}**

**…"**

## Constraints

* The name of each fish in the pool will always be unique.
* Each fish will have a different number of fins.
* The fins of a fish and the size of the aquarium will always be positive numbers.
* You will always be given fish added before receiving the method for its manipulation.

### Examples

|  |
| --- |
| Sample code usage |
| package aquarium;  public class Main {  public static void main(String[] args) {  // Initialize Aquarium Aquarium aquarium = **new** Aquarium(**"Ocean"**, 5, 15);  // Initialize Fish Fish fish = **new** Fish(**"Goldy"**, **"gold"**, 4);  // Print Fish System.***out***.println(fish.toString());   //Fish: Goldy  //Color: gold  //Number of fins: 4  // Add Fish aquarium.add(fish);  // Find Fish  aquarium.findFish(**"Goldy")**;  // Get Fish in Pool  aquarium.getFishInPool();   // Remove Fish System.***out***.println(aquarium.remove(**"Goldy"**)); // true  Fish secondFish = **new** Fish(**"Dory"**, **"blue"**, 2);  Fish thirdFish = **new** Fish(**"Nemo"**, **"orange"**, 5);   // Add fish aquarium.add(secondFish);  aquarium.add(thirdFish);   // Print Aquarium report System.***out***.println(aquarium.report());   //Aquarium Info:  //Aquarium: Ocean ^ Size: 15  //Fish: Dory  //Color: blue  //Number of fins: 2  //Fish: Nemo  //Color: orange  //Number of fins: 5  }  } |