# 01.Masterchef

**You can test your solutions in** [**Judge**](https://judge.softuni.org/Contests/Practice/Index/3020#0)**.**

*Do you believe you have what it takes to become the new Masterchef? For tonight’s challenge you have to use the ingredients in your basket to prepare 4 excellent dishes.*

First you will receive a sequence of **integers**, representing the number of ingredients in a single basket. After that you will be given another sequence of **integers** - the freshness level of the ingredients.

Your task is to **cook** them so you can impress the judges. The names of the dishes are listed in the table below with the exact freshness level.

|  |  |
| --- | --- |
| **Dish** | **Freshness Level needed** |
| Dipping sauce | 150 |
| Green salad | 250 |
| Chocolate cake | 300 |
| Lobster | 400 |

To cook a dish, you have to take the **first** **ingredient** **value** and the **last freshness level value**. The total freshness level is calculated by their **multiplication**.

* If the product of this operation **equals** one of the levels described in the table, you make the dish and **remove both** ingredient and freshness value.
* **Otherwise** you should **remove the freshness level**, then **increase** the ingredient value by **5**
  + **Remove the ingredient from the collection and add it again in last place, already increased by 5**.
* In case you have an ingredient with value **0** you have to **remove** it and continue cooking.

You need to **stop** **cooking** **only** when you **run out of ingredients** **or freshness level** values.

Your task is considered **successful** if you make at least **four** dishes - **one of each type**.

## Input

* The first line of input will represent the ingredients' values - **integers**, separated by **single space**
* On the second line you will be given the freshness values - **integers** again, separated by **single space**

## Output

* On the first line of output - print if you've succeeded in preparing the dishes
* "**Applause! The judges are fascinated by your dishes!**"
* "**You were voted off. Better luck next year.**"
* On the next output line - print the **sum** of the ingredients **only** if they are left **any** in the format: "**Ingredients left: {ingridientsSum}"**
* On the last few lines you have to print the dishes you have made ordered **alphabetically, but only the ones that were made at least once** in the format:

" **# {dish name} --> {amount}**"

## Constraints

* All of the ingredients' values and freshness level values will be **integers** in range **[0, 100]**
* We can have **more than one** cooked dish of the types specified in the table above

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| **10 10 12 8 10 12**  **25 15 50 25 25 15** | Applause! The judges are fascinated by your dishes!  **#** Chocolate cake **--> 2**  **#** Dipping sauce **--> 2**  **# Green salad --> 1**  **# Lobster --> 1** | First you take the **first** ingredient and the **last** freshness level value and **multiply** them - the result is 150 so we **make** Dipping sauce. Next we have product of 250 and the Green salad is **ready**. Then we **cook** the Chocolate cake by multiplying 12 and 25. The product of next ingredient value and freshness level value is 400 and we **make** Lobster. Next pair is 10 and 15, we multiply them and make one more Dipping sauce. The last multiplication of 12 and 25 equals 300 and we make one more Chocolate cake. There are **no more ingredients and freshness values** so we stop cooking, but we have **one of each** dish types and print the **proper** message. |
| **12 20 0 6 19**  **12 12 25** | **You were voted off. Better luck next year.**  **Ingredients left: 55**  **# Chocolate cake --> 1** | **The first pair is 12 and 25, we cook Chocolate cake and remove both of them.**  **Next we take 20 and 12 - the product is 240 - we **can't cook any meal**, so **we remove** the freshness level value, increase the ingredient value with 5 and add it back in last place.** **The next ingredient has value 0 - we **remove it and continue.****  **The next pair is 6 and 12 - again we can't make anything. After that we don't have more freshness level values, so **we stop cooking**.** **The rest of the ingredients are 19, 25, 11 with sum of 55.** |

# 02. The Battle of the Five Armies

**You can test your solutions in** [**Judge**](https://judge.softuni.org/Contests/Practice/Index/3088#1)**.**

The five armies refer to the Goblins, Wolves, Elves, Men and Dwarves join forces to fight off the orcs.

In order to do that, they have to go through the land and fight off the orcs. If they successfully reach Mordor, they will win the fight against the evil.

A standard map of the Middle World looks like this:

|  |  |
| --- | --- |
| **The Middle World** | **Legend** |
| ------M--- -------O-- --O------- ---------- -----A---- | **A** 🡺 The **Army**, the player character  **O**🡺 Orcs**, enemy**  **M** 🡺 Mordor  - 🡺 **Empty space** |

Each turn proceeds as follows:

* **First**, Orcs **spawn** on the given index.
* **Then, the army** moves in a direction, which **decreases** their armor by 1.
  + It can be “**up**”, “**down**”, “**left**”, “**right**”
  + If **the army** tries to move **outside** of the field, they **don't** move but **still** has their energy **decreased**.
* If an enemy is on the **same cell** where the army moves, the army fights him, which **decreases** their armor by 2. If the army’s armor **drops** at 0 or below, they **die** and you should mark this position with ‘**X**’.
* If the army **kills** the enemy successfully, the enemy **disappears**.
* If the army reaches the **index** where **Mordor** is, **they win the war** (disappear from the field)**, even if their armor is 0 or below.**

## Input

* On the **first line** of input, you will receive **e** – **the armor the army** has.
* On the **second line** of input, you will receive **n** – the **number of rows** the map of the Middle World will consist of.  
  Range: **[5-20]**
* On the next **n lines**, you will receive how each row looks.
* Then, **until** the army dies, or reaches the throne, you will receive a **move command** and **spawn row and column**.

## Output

* If the army is **runs out of armor**, print "The army was defeated at {row};{col}."
* If the army **reaches Mordor**, print "The army managed to free the Middle World! Armor left: {armor}"
* Then, in all cases, **print** the **final state of the map** on the **console**.

## Constraints

* The map will always be **rectangular**.
* The army will **always** run out of armor or reach Mordor.
* There will be **no case** with spawn on **invalid** index.
* There will be **no case** with **two enemies on the same cell**.
* There will be **no case** with enemy **spawning** on the index **where the army or Mordor is**.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 100  5  --M--  -----  -----  -----  --A--  up 3 0  up 3 1  up 3 2  up 3 3 | The army managed to free the Middle World! Armor left: 96  -----  -----  -----  OOOO-  ----- | Turn 1: An enemy spawns at [3;0], the army moves to [3;2], their armor decreases by 1.  Turn 2: An enemy spawns at [3;1], the army moves to [2;2], their armor decreases by 1.  Turn 3: An enemy spawns at [3;2], the army moves to [1;2], their energy decreases by 1.  Turn 4: An enemy spawns at [3;3], the army moves to [0;2], their energy decreases by 1, but they also move to the index where Mordor is – they win the war |
| 3  5  --M--  -----  -----  -----  --A--  up 3 2 | The army was defeated at 3;2.  --M--  -----  -----  --X--  ----- |  |

# 03. Street Racing

**You can test your solutions in** [**Judge.**](https://judge.softuni.org/Contests/Practice/Index/3088#2)

*You and your friends decide to organize some illegal street races. Since you are a programmer, you have the task to develop software for organizing safe and fair races.*

## Preparation

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

## Problem description

Your task is to create a race in which participate different cars.

### Car

First, write a C# **class**, called **Car** with properties:

* **Make: string**
* **Model: string**
* **LicensePlate: string**
* **HorsePower: int**
* **Weight: double**

The **constructor** of Car class should receive **make, model, licensePlate, horsePower and weight**.

The class should also have the following methods:

* Override **ToString()** method in the format:

**"Make: {Make}**

**Model: {Model}**

**License Plate: {LicensePlate}**

**Horse Power: {HorsePower}**

**Weight: {Weight}"**

### Race

Next step is to write **Race** class that has a **collection** of type **Car** with corresponding **unique** **License Plate** of a Car. The name of the collection should be **Participants**. All the entities of the **Participants** collection have the **same** properties. The **Race** has also some additional properties:

* **Name: string**
* **Type: string**
* **Laps: int**
* **Capacity: int -** the maximum allowed number of participants in the race
* **MaxHorsePower: int -** the maximum allowed Horse Power of a Car in the Race

The **constructor** of the Race class should receive **name, type, laps, capacity** and **maxHorsePower**.

Implement the coming features:

* Getter **Count** - returns the count of the currently enrolled participants
* Method **Add(Car car)** - adds the entity **if** there **isn't** a Car with the same **License plate** and **if** there is enough space in terms of race **capacity** and if the car meets the **maximum horse power** requirment of the race.
* Method **Remove(string licensePlate)** - removes a Car from the race with the given **License plate**, if such **exists** and returns **bool** if the deletion is successful.
* Method **FindParticipant(string licensePlate)** - returns a **Car** with the given License plate. If it doesn't exist, return **null**.
* Method **GetMostPowerfulCar() –** returns the **Car** with most **HorsePower**. If there are no Cars in the Race, method should return null.
* Method **Report()** - returns information about the Race and the Cars participating it in the following format:

**"****Race: {Name} - Type: {Type} (Laps: {Laps})**

**{Car1}**

**{Car2}**

**… "**

## Constraints

* The License plate of each Car in the race will always be unique
* The HorsePower of a Car and the MaxHorsePower of the Race will always be positive numbers
* There will not be a case where two Cars have the same HorsePower
* You will always be given Car added before receiving method for its manipulation

### Examples

|  |
| --- |
| Sample code usage |
| //Sample Code Usage:  //Initialize Race  Race race = new Race("RockPort Race", "Sprint", 1, 4, 150);  //Initialize Car  Car car = new Car("BMW", "320ci", "NFS2005", 150, 1450);  //Print car  Console.WriteLine(car.ToString());  //Make: BMW  //Model: 320ci  //License Plate: NFS2005  //Horse Power: 150  //Weight: 1450  //Add car  race.Add(car);  //Remove car  Console.WriteLine(race.Remove("NFS2005")); // True  Car carOne = new Car("BMW", "320cd", "NFS2007", 150, 1350);  Car carTwo = new Car("Audi", "A3", "NFS2004", 131, 1300);  //Add cars  race.Add(carOne);  race.Add(carTwo);  //GetMostPowerfulCar  Console.WriteLine(race.GetMostPowerfulCar());  //Make: BMW  //Model: 320cd  //License Plate: NFS2007  //Horse Power: 150  //Weight: 1350  //Print Race report  Console.WriteLine(race.Report());  //Race: RockPort Race - Type: Sprint (Laps: 1)  //Make: BMW  //Model: 320cd  //License Plate: NFS2007  //Horse Power: 150  //Weight: 1350  //Make: Audi  //Model: A3  //License Plate: NFS2004  //Horse Power: 131  //Weight: 1300 |

## Submission

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