# Blacksmith



*You are the most well-known blacksmith on Middle Earth. What makes your swords so good is the perfect ratio between* *steel and carbon*, *which makes them extremely sharp and durable.*

First, you will be given **a sequence representing steel**. Afterward, you will be given another **sequence representing carbon**.

You should start from the **first steel** and try to mix it with the **last carbon.** If the **sum** of their values is **equal** to **any of the swords in the table below** you should forge the **sword corresponding** to the **value** and **remove** **both** the **steel** and the **carbon**. Otherwise, **remove only the steel**, **increase** the **value** of the **carbon by 5** and **insert** it **back** into the **collection**. You need to **stop** forging when you have **no more steel** or **carbon left**.

|  |  |
| --- | --- |
| **Sword** | **Resources needed** |
| Gladius | 70 |
| Shamshir | 80 |
| Katana | 90 |
| Sabre | 110 |

Forge as **many swords as possible.**

### Input

* On the **first line**, you will receive the steel, **separated** by a **single space (**" "**)**.
* On the **second line**, you will receive the carbon, **separated** by a **single space (**" "**)**.

### Output

* On the **first** line of output depending on the result:
  + If at least one sword was forged: "**You have forged {totalNumberOfSwords} swords.**"
  + If no sword was forged: "**You did not have enough resources to forge a sword.**"
* On the **second** line - print all steel you have left:
  + If there are no steel: "**Steel left: none**"
  + If there are steel: "**Steel left: {steel1}, {steel2}, {steel3},** **(…)**"
* On the **third** line - print all carbon you have left:
  + If there are no carbon: "**Carbon** **left: none**"
  + If there are carbon: "**Carbon** **left: {carbon1}, {carbon2}, {carbon3},** **(…)"**
* Then**,** you need to print **only the swords that you manage to forge** and how many **of them**, **ordered** **alphabetically**:
  + **"Sabre: {amount}"**
  + **"Katana: {amount}"**
  + **"Shamshir: {amount}"**
  + **"Gladius: {amount}"**

### Constraints

* All of the given numbers will be valid resources in the range **[0, 130]**.

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **40 50 70 120 10 20**  **30 20 30 20 30 50** | **You have forged 3 swords.**  **Steel left: none**  **Carbon left: 45, 20, 30**  **Katana: 2**  **Shamshir: 1** | We start by taking **40 (steel) + 50 (carbon) = 90**. The first sword is forged "**Katana**" and we remove both materials.  Next **50 (steel) + 30 (carbon) = 80**, "**Shamshir** " is forged and we remove both materials.  Next **70 (steel) + 20 (carbon) = 90**, "**Katana**" is forged and we remove both materials.  Next **120 (steel) + 30 (carbon) = 150**, no sword could be made, we remove **steel**, increase **carbon** by 5 and insert back **(35)** into the collection.  Next **10 (steel) + 35 (carbon) = 45**, no sword could be made, we remove **steel**, increase **carbon** by 5 and insert back **(40)** into the collection.  In the last iteration **20 (steel) + 40 (carbon) = 65** no sword could be made, we remove **steel**, increase **carbon** by 5 and insert back **(45)** into the collection.  We’ve menage to forge **3 swords** and left with **2 pieces (45, 20, 30)** of carbon. |
| **10 5 30**  **30 20 10** | **You did not have enough resources to forge a sword.**  **Steel left: none**  **Carbon left: 25, 20, 30** |  |

# 2. Armory



*You have forged many swords, so now you start selling them. There are lots of customers who want to buy your blades, but you do not want to bargain for every single sword and decide to sell them to the king. The king sends an army officer to pick swords for his army. Your armory is huge, so you need to guide the army officer though.*

You will be given an integer **n** for the **size** of the armory with a **square** shape. On the next **n** lines, you will receive the **rows** of the armory. The army officer will be placed in a **random position**, marked with the letter '**A**'. On random positions, there will be **swords**, marked with a **single digit (the price of the sword)**. There **may** also be **mirrors**, the **count** will be either **0** or **2** and they are **marked** with the **letter** - '**M**'. **All of the empty positions** will be marked with **'-'**.

Each turn, you will tell the army officer which direction he should move. Move commands will be: "**up**", "**down**", "**left**", "**right**". If the army officer **moves** to a **sword**, he **buys the sword** **for a price** **equal** to the **digit** **there** and the sword **disappears**. If the army officer moves to a **mirror**, he teleports on the **position** of the **other mirror,** and then **both** mirrors **disappear**. If you guide the army officer **out of the** armory, he leaves with the swords that he has bought. In advance, you negotiate with the king, that he'll buy **at least 65 gold coins worth of blades.**

**The program ends when the army officer buys blades for at least 65 gold coins, or you guide him out of the armory.**

### Input

* On the first line, you are given the integer **n** – the size of the matrix (armory).
* The **next n lines** hold the values for every **row**.
* On each of the next lines, you will get a move command.

### Output

* On the first line:
  + If the army officer leaves the armory, print: "**I do not need more swords!**"
  + If the army officer fulfills the initial deal, print: "**Very nice swords, I will come back for more!**"
* On the second line print the profit you’ve made: "**The king paid {amount} gold coins.**"
* At the end print the final state of the matrix (armory).

### Constraints

* The size of the **square** matrix (armory) will be between **[2…10].**
* There will **always** be **0** or **2** mirrors, marked with the **letter** - '**M**'.
* The army officer’s position will be marked as '**A**'.
* There will be always two output scenarios: the army officer leaves or bays swords worth at least **65 gold** coins.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 4  A9--  -M--  ----  M---  right  down  left | I do not need more swords!  The king paid 9 gold coins.  ----  ----  ----  ---- | The first command is "**right**". The army officer moves to **a sword, and buys it.**  -А--  -M--  ----  M---  **Nex command** is "**down**" the army officer steps on the mirror and teleports to the bottom left corner of the armory.  ----  ----  ----  А---  The last command is "**left**". The army officer leaves the armory. |
| 6  A99---  99----  99996-  ---9--  ---9--  -6-6--  right  right  down  left  left  down  right  right  right | Very nice swords, I will come back for more!  The king paid 72 gold coins.  ------  ------  ---A6-  ---9--  ---9--  -6-6-- | Here we have **no** mirrors and lots of swords in the armory.  The army officer buys enough swords to fulfill the deal.  All swords which were sold disappear and we can see where the army officer is located when the deal is fulfilled (2, 3). |

# 3. Easter Basket



*It’s Easter and we have a basket full of Easter eggs. We want the basket to look beautiful, so we have to do some changings.*

**Preparation**

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

**Pay attention to name the package easterBasket, all the classes, their fields and methods the same way they are presented in the following document. It is also important to keep the project structure as described.**

**Problem description**

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

First, write a class **Egg** with the following properties:

* **color: String**
* **strength: int**
* **shape: String**

The class **constructor** should receive **color, strength and shape**. You need to create the appropriate **getters and setters**. Override the **toString()** method in the following format:  
**"{color} egg, with {strength} strength and {shape} shape."**

**Next**, write a class **Basket** that has **data** (a List which stores the entity **Egg**). All entities inside the repository have the **same properties**. Also, the **Basket** class should have those **properties**:

* **material: String**
* **capacity: int**

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

* **List<Egg> data** - **collection** that holds added eggs
* **Method addEgg(Egg egg)** – **adds** an **entity** to the data **if** **there** **is** **room** for it
* **Method removeEgg(string color)** – removes an egg by **given color,** if such **exists**, and **returns boolean** (true if it is removed, otherwise – false)
* **Method getStrongestEgg()**– **returns the strongest egg**
* **Method getEgg(string color)** – **returns** the **egg** with the **given color**
* **Method getCount** – **returns** the **number** of **eggs**
* **Method report()** – **returns** a **string** in the following **format** (print the eggs in **order of appearance**):
  + **"{material} basket contains:  
    {Egg1}  
    {Egg2}  
    (…)"**

**Constraints**

* The **color** and **strength** of the eggs will be **always unique**.
* You will always have an egg added before receiving methods manipulating the Basket’s eggs.

**Examples**

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

|  |
| --- |
| **Sample code usage** |
| //Initialize the repository (Basket)Basket basket = **new** Basket(**"**Wood**"**, 20); //Initialize entity (Egg)Egg egg = **new** Egg(**"**Red**"**, 10, **"**oval**"**); //Print EggSystem.***out***.println(egg); //Red egg, 10 strength, oval shape//Add Eggbasket.addEgg(egg);  //Remove EggSystem.***out***.println(basket.removeEgg(**"**Pink**"**)); //FalseEgg secondEgg = **new** Egg(**"**Green**"**, 9, **"**pointy**"**);  //Add Eggbasket.addEgg(secondEgg);  //Get strongest eggEgg strongestEgg = basket.getStrongestEgg();  System.***out***.println(strongestEgg);  //Get eggEgg getEgg = basket.getEgg(**"**Green**"**); //Green egg with 9 strength, pointy shapeSystem.***out***.println(getEgg);   System.***out***.println(basket.report()); //Wood basket contains: //Red egg, with 10 strength and oval shape. //Green egg, with 9 strength and pointy shape. |

**Submission**

Submit **single .zip file**, containing **easterBasket** package, **with the classes inside** (**Basket** and **Egg** and the **Main** **class)**, there is no specific content required inside the **Main** class e. g. you can do any kind of local testing of you program there. However, there should be **main(String[] args)** method inside.