# Energy Drinks

You can test your solutions in [Judge](https://judge.softuni.org/Contests/Practice/Index/3624#0)

*Your friend Stamat is working on a new AI program. Like every irresponsible teenager, he programs all night and, of course, drinks a lot of energy drinks. Stamat***'***s friends are concerned about him and want you to create a program that tells him when to stop the energy drinks and start drinking water.*

On the first line, you will receive a sequence of **numbers** representing **milligrams of caffeinе**. On the second line, you will receive another sequence of **numbers** representing **energy drinks**. It is important to know that the **maximum caffeine** Stamat can have for the night is **300 milligrams,** and his **initial is always 0**.

To **calculate the caffeine in the drink take the** **last milligrams of caffeinе** and the **first energy drink, and multiply them**. Then, **compare the result** with the **caffeine Stamat drank**:

* If the sum of the **caffeine** in the drink and **the caffeine that Stamat drank** **doesn't exceed 300 milligrams**, **remove** **both** the **milligrams of caffeinе** and the **drink** from their sequences. Also, **add the caffeine** to Stamat**'**s total caffeine.
* If Stamat is **about to** **exceed** his maximum caffeine per night, **do not add** the caffeine to Stamat’s total caffeine. **Remove the** **milligrams of caffeinе** and **move the drink to the end of the sequence**.Also, **reduce the current caffeine** that Stamat has taken **by 30** (Note: Stamat's caffeine cannot go below 0).

**Stop calculating** when you are **out of drinks** or **milligrams of caffeine**.

For more clarification, see the examples below.

### Input

* In the **first line,** you will be given a **sequence of the milligrams of caffeinе** - **integers** separated by comma and space **", "** in the range **[1, 50]**
* In the **second line,** you will be given a **sequence of energy drinks** - **integers** separated by comma and space **", "** in the range **[1, 300]**

### Output

* On the first line:
  + If **Stamat hasn't drunk all the energy drinks**, print **the remaining ones** separated by a comma and a space **", "**:
    - **"Drinks left: { remaining drinks separated by ", " }"**
* If Stamat **has** **drunk all the energy drinks**, print:
  + **"At least Stamat wasn't exceeding the maximum caffeine."**
* On the next line, print:
  + **"Stamat is going to sleep with { current caffeine } mg caffeine."**

### Constraints

* You will always have **at least one element in each sequence at the beginning**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 34, 2, 3  40, 100, 250 | Drinks left: 100, 250  Stamat is going to sleep with 60 mg caffeine. |
| **Comment** | |
| 1) Take the last milligrams of caffeine (3) and multiply them by the first energy drink (40). The result(120) doesn’t exceed the caffeine limit per day (300), so we can add it to Stamat's caffeine. Remove both items from their sequences. Stamat can accept 180 miligrams of caffeine more.  2) Take the next mg of caffeine (2) and multiply them by the next energy drink (100). The result is 200 and if he takes the drink, he will exceed the caffeine limit per day. We remove the mg of caffeine (2) and place the drink (100) at the end of the sequence ("250, 100"). Then, decrease Stamat's caffeine by 30 (Stamat's caffeine becomes 90). Stamat can accept 210 miligrams of caffeine more.  3) Take the next mg of caffeine (34) and multiply them by the next energy drink (250). The result(8500) is above 210, so we remove the mg of caffeine (34) and place the drink (250) at the end of the sequence ("100, 250"). Then, decrease Stamat's caffeine by 30 (Stamat's caffeine becomes 60).  4) Stamat slept with 60 mg of caffeine. | |
| **Input** | **Output** |
| 1, 16, 8, 14, 5  27, 23 | At least Stamat wasn't exceeding the maximum caffeine.  Stamat is going to sleep with 289 mg caffeine. |
| **Input** | **Output** |
| 1, 23, 2, 1, 42, 22, 7, 14  51, 100, 3, 7 | At least Stamat wasn't exceeding the maximum caffeine.  Stamat is going to sleep with 264 mg caffeine. |

# 2. Rally Racing

You can test your solutions in [Judge](https://judge.softuni.org/Contests/Practice/Index/3624#1)

*It's time for one of the biggest races in the world, Paris-Dakar. The organizers of the event want you to do a program that helps them track the cars through the separate stages in the event.*

On the first line, you will be given an **integer N**, which represents the **size of a square matrix**. On the second line you will receive the **racing number** of the tracked race car.

On the **next N lines** you will be given the rows of the matrix (**string sequences**, separated by whitespace), whichwill be representing the **race route**. The tracked race **car** **always** starts with **coordinates [0, 0].** Thеre will be a **tunnel** somewhere across the race route. If the race car runs into the **tunnel** , the **car goes through** it and **exits** at **the other end**. There will be **always two positions** marked with **"T"(tunnel)**. The **finish line** will be marked with **"F".** All other positions will be marked with **"."**.

Keep track of the **kilometers passed**. Every time the race car receives a direction and moves to the **next position** of the race route, it **covers 10 kilometers**. If the car **goes through the tunnel**, it **covers** NOT 10, but **30 kilometers**.

On **each line**, after the matrix is given, you will be receiving **the directions** for the race car.

* left
* right
* up
* down

The race car starts moving across the race route:

* If you receive **"End"** command, before the race car manages to reach the finish line, the car is disqualified and the following output should be printed on the Console: **"Racing car {racing number} DNF."**
* If the race car comes across a position marked with **"."**. The car **passes 10 kilometers** for the current move and waits for the next direction.
* If the race car comes across a position marked with **"T"** this is the **tunnel**. The race car goes through it and **moves to the other position marked with "T"** (the other end of the tunnel).The car **passes 30 kilometers** for the current move. The tunnel stays behind the car, so **the race route is clear**, and **both** the positions marked with **"T"**, **should be marked with "."**.
* If the car **reaches the finish line - "F"** position, the race is over. The tracked race car manages to finish the stage and the following output should be printed on the Console: **"Racing car {racing number} finished the stage!".** Don’t forget that the car has covered another 10 km with the **last** move.

### Input

* On the first line you will receive N - the size of the square matrix (race route)
* On the second line you will receive the racing number of the tracked car
* **On the next N lines,** you will receive the **race route** (**elements** will be **separated by a space**).
* On the following lines, you will receive directions **(left, right, up, down).**
* On the last line, you will receive the command **"End"**.

### Output

* If the racing car has reached the finish line before the **"End"** command is given, print on the Console: **"Racing car {racing number} finished the stage!"**
* If the **"End"** command is given and the racing car **has not reached the finish line yet**, the **race ends** and the following message is printed on the Console: **"Racing car {racing number} DNF."**
* On the second line, print the distance that the tracked race car has covered: **"Distance covered {kilometers passed} km."**
* At the end, mark the **last known position of the race car with** **"C" and** print the **final state of the matrix** (race route). If the race car hasn’t gone through the tunnel, the tunnel exits **should be visualized** in the final state of the matrix. The **row elements** in the output matrix **should NOT be separated** by a whitespace.

### Constraints

* The directions will always lead to coordinates in the matrix.
* There will always be two positions marked with **"T"** , representing the tunnel in the race route.
* The size of the **square** matrix (**race route**) will be between **[4…10].**

### Еxamples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 5  01  . . . . .  . . . T .  . . . . .  . T . . .  . . F . .  down  right  right  right  down  right  up  down  right  up  End | Racing car 01 finished the stage!  Distance covered 80 km.  .....  .....  .....  .....  ..C.. | The race car starts moving from position[0,0].  The first command is down, so the moving direction is down. The race car is in position[1,0].  Next three commands are right, so the race car comes across the tunnel – "T". The current car position is [1,3]. Swap the "T" with "." The race car goes through the tunnel, so its next position is [3,1]. Swap the "T" with "."  Next direction is down, so the race car position is [4,1].  Next direction is right, so the race car position is [4,2].  The race car reaches the finish line before the "End" command. So it manages to finish the stage. The remaining directions will be ignored and no more moves are going to be executed. |
| 10  45  . . . . . . . . . .  . . T . . . . . . .  . . . . . . . . . .  . . . . . . . . . .  . . . . . . . . . .  . . . . . . . . . .  . . . . . . F . . .  . . . . . . . . . .  . . . . . . . . . .  . . . . . . . T . .  right  down  down  right  up  left  up  up  End | Racing car 45 DNF.  Distance covered 100 km.  ..........  ..........  ..........  ..........  ..........  ..........  ......F...  ......C...  ..........  .......... |  |

# Computer Architecture

You can test your solutions in [Judge](https://judge.softuni.org/Contests/Practice/Index/3624#2)

*Here you are and you have already successfully passed several courses at SoftUni, congratulations. But have you ever wondered about how exactly the hardware of a computer is designed? This problem description will give you a peek into the architecture of a computer system.*

## 1. Preparation

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

## 2. Problem description

Your task is to create a computer repository that stores CPU components by creating the classes described below.

**CPU (Central Processing Unit)**

You are given a class **CPU,** create the following properties:

* **Brand - string**
* **Cores - int**
* **Frequency - double**

The class **constructor** should receive **brand, cores, and frequency**.

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

**"{brand} CPU:**

**Cores: {cores}**

**Frequency: {frequency} GHz"**

**Note: Format the Frequency to the first digit after the decimal point!**

**Computer**

**Next**, you are given a class **Computer** that has **a Multiprocessor** (a collection that stores **CPU** entities). All entities inside the collection have the **same fields**. Every **Computer** will have **Capacity** of the motherboard, and **adding new CPU will be limited** by the Capacity. Also, the **Computer** class should have the following **properties**:

* **Model - string**
* **Capacity – int**
* **Multiprocessor – List<CPU>**

The class **constructor** should receive the **model** and **capacity**, also it should initialize the **multiprocessor** with a new instance of the collection.

Implement the following features:

* Getter Count - **returns** the **number** of CPUs
* Method Add(CPU cpu) - **adds** an **entity** to the multiprocessor **if** **there** **is** **room** for it. If there is no room for another CPU, skip the command
* Method Remove(string brand) - removes a CPU by a **given brand. I**f such **exists**, returns **true**, otherwise, returns **false**.
* Method MostPowerful() - returns **the most powerful** CPU(the CPU with the **highest frequency**)
* Method **GetCPU(string brand)** – returns the CPU with the **given brand**. If there is no CPU, meeting the requirements, return **null**
* Method **Report()** - **returns** a **String** in the following **format:**
  + **"CPUs in the Computer {model}:  
    {CPU1}  
    {CPU2}  
    (…)**"

## Constraints

* The **models** of the computers will be **always unique**.
* The **capacity** of the computer will always be with **positive values**.
* The **brand** of the CPUs will be **always unique**.
* The **cores** of the CPUs will always be with **positive values**.
* The **frequency** of the CPUs will always be with **positive values**.
* You will always have a CPUs added before receiving methods manipulating the Computer's multiprocessor.

## Examples

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

|  |
| --- |
| Sample code usage |
| *// Initialize the repository* Computer computer = **new** Computer(**"Gaming Serioux"**, 4);  *// Initialize entity* CPU cpu = **new** CPU(**"AMD Ryzen 5"**, 6, 3.7);  *// Print CPU* Console.WriteLine(cpu);  // AMD Ryzen 5 CPU:  // Cores: 6  // Frequency: 3.7 GHz  *// Add CPU* computer.Add(cpu);  *// Remove CPU* Console.WriteLine(computer.Remove(**"Intel Core i5"**));  // False  CPU secondCPU = **new** CPU(**"Intel Core i7"**, 8, 4);  CPU thirdCPU = **new** CPU(**"Intel Core i5"**, 8, 3.9);  *// Add CPU* computer.Add(secondCPU);  computer.Add(thirdCPU);  CPU mostPowerful = computer.MostPowerful();  Console.WriteLine (mostPowerful);  // Intel Core i7 CPU:  // Cores: 8  // Frequency: 4.0 GHz  CPU receivedCPU = computer.GetCPU(**"Intel Core i5"**);  Console.WriteLine(receivedCPU);  // Intel Core i5 CPU:  // Cores: 8  // Frequency: 3.9 GHz  Console.WriteLine(computer.Count);  // 3  Console.WriteLine(computer.Remove(**"Intel Core i5"**));  // True  Console.WriteLine(computer.Report()); // CPUs in the Computer Gaming Serioux:  // AMD Ryzen 5 CPU:  // Cores: 6  // Frequency: 3.7 GHz // Intel Core i7 CPU:  // Cores: 8  // Frequency: 4.0 GHz |

## Submission

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