# 

# Climb the Peaks

You can test your solutions in [Judge](https://judge.softuni.org/Contests/Practice/Index/3625#0)

*Alex is a vlogger and he wants to make videos, climbing the five highest peaks in Pirin mountain in just* ***one week****. He will take his video set, a tent and his* ***backpack*** *in the mountain. The backpack fits* ***food portions*** *for one week, exactly. His* ***daily stamina*** *is also limited. Your task is to trace his adventure and to create a post for his profile @alaroundtheworld, at the end of the journey.*

You will have to keep information for **all the conquered peaks**, if any.

Every day, Alex will **use one portion** of his **daily food supplies** and will **exhaust one**  of his **daily stamina**.

First, you will be given **a sequence of numbers, representing the quantities of the daily portions** of food supplies in his backpack.

Afterwards, you will be given another **sequence of numbers, representing the quantities of the daily stamina** he will have on his disposal for the next **seven days**.

You have to **sum** the **quantity of the** **last daily food portion** with the **quantity of the first daily stamina**. He will start climbing **from the first** peak in the table below **to the last** one.

* If the **sum is equal or greater** than the corresponding **Mountain Peak’s Difficulty level from the table below**, it means that the **peak is conquered**. In this case, you should **remove both quantities from the sequences** and **continue** with the **next ones** towardsthe **next peak**.
* If the **sum** is less than the corresponding **Mountain Peak’s Difficulty level** from the table below, the **peak remains unconquered**. You should **remove both quantities from the sequences.** Alex will have to sleep in his tent. On the next day he will try **the same peak once again**.

|  |  |
| --- | --- |
| **Mountain Peaks** | **Difficulty level** |
| Vihren | 80 |
| Kutelo | 90 |
| Banski Suhodol | 100 |
| Polezhan | 60 |
| Kamenitza | 70 |

Alex will try to conquer as much peaks as he can in seven days. If he **eats all the food** **supplies** and **exhausts all his stamina** before the last day of his journey, the adventure is over. If he manages to climb **all the peaks**, the journey ends and the output is printed on the Console.

Finally, **print** on the Console **all the conquered peaks**(in the order of climbing).

### Input

* On the **first line**, you will receive the **food portions** quantities, **separated** by **comma and a** **single space (',** **')**.
* On the **second line**, you will receive the **stamina** quantities, **separated** by **comma and a** **single space (', ')**.

### Output

* On the first line – print whether Alex managed to reach his goal and climb all the peaks in his list:
  + If he managed to conquer all: "**Alex did it! He climbed all top five Pirin peaks in one week -> @FIVEinAWEEK**"
  + If he didn't manage to climb all of the peaks: "**Alex failed! He has to organize his journey better next time -> @PIRINWINS**"
* Then, in either case, if Alex **fails or succeed** in completing the climbing adventure, you should print all **conquered peaks** (in the order of climbing), if there are any:

"Conquered peaks:

{peak1}

{peak2}

…

{peakn}"

* If there are **no conquered peaks do not print** this message.

### Constraints

* All of the given numbers will be valid integers in the range **[0…100]**.
* Do not use "**\r\n**" for a new line.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| **40, 40, 40, 40, 40, 40, 40**  **40, 50, 60, 20, 30, 5, 2** | Alex did it! He climbed all top five Pirin peaks in one week -> @FIVEinAWEEK  Conquered peaks:  Vihren  Kutelo  Banski Suhodol  Polezhan  Kamenitza | We start by taking the last daily portion quantity (**40**) and the first stamina quantity (**40**). Their **sum** is **40 + 40 = 80**. After that, we check if the sum is equal or greater than the **first peak’s difficulty level**. The **sum** of the food portion and the stamina for the day equals to **the peak’s difficulty level**, so the **peak is conquered**. We **remove both quantities** from the sequences.  We contunue with the next peak, where we have also enough food and stamina to conquer it. After we conquer all five peaks, the program ends, and we print the final post on the Console. After that we print all conquered peaks, every peak on a new line. |
| **10, 20, 34, 26, 12, 10, 45**  **30, 28, 17, 17, 13, 10, 10** | Alex failed! He has to organize his journey better next time -> @PIRINWINS | We start by taking the last daily portion quantity (**45**) and the first stamina quantity (**30**). Their **sum** is **45 + 30 = 75**. After that, we check if the **sum** is equal or greater than the **first peak’s difficulty level**. The **sum** is not enough for the peak to be conquered. Alex will have to sleep in the tent. We **remove both quantities** from the sequences.  On the **next day** he will try the **same peak** once again, so he will need a **sum of 80**, from the food portion and the daily stamina. The sum is 10 + 28 = 40 and it is not enough. Alex will have to sleep in his tent once again. We **remove both quantities** from the sequences.  He will not be able to climb **any of the peaks**, so we should print on the Console a message for an **unsuccessful adventure**. There won’t be **any conquered peaks**, so we will print only the **status message on the Console.** |

# 

# 02. Navy Battle

You can test your solutions in [Judge](https://judge.softuni.org/Contests/Practice/Index/3625#1)

*1914, September 22 – German submarine* [*U-9*](https://en.wikipedia.org/wiki/SM_U-9)[*sinks three unescorted British armored cruisers*](https://en.wikipedia.org/wiki/Action_of_22_September_1914)[*HMS*Aboukir](https://en.wikipedia.org/wiki/HMS_Aboukir_(1900))*,* [*HMS*Hogue](https://en.wikipedia.org/wiki/HMS_Hogue_(1900)) *and* [*HMS*Cressy](https://en.wikipedia.org/wiki/HMS_Cressy_(1899)) *in approximately one hour. Imagine that they had the technology to make themselves a navigational program for the submarine and you are chosen to implement the logic. Navigate U-9 through the battlefield, find and sink the British cruisers in the dark night, avoiding the floating mines all over the North Sea.*

You will be given an integer **n** for the size of the **battlefield** (square shape). On the next **n** lines, you will receive the rows of the **battlefield**. The submarine will start at a **random** position, marked with the letter '**S**'. The submarine surveys the surrounding area through its periscope, so it has to climb up to periscope depth, where it might run across naval **mines**.

When the submarine receives direction, it goes deep and moves **one position toward the given direction**. On each turn, you will be guiding the submarine and giving it the **direction**, in which it should move. The commands will be "**up**", "**down**", "**left**" and "**right**".

When a **new position is reached**, the submarine climbs up to periscope depth to search for a cruiser:

* If a position with '**-**' (dash) is reached, it means that the field is empty and the submarine awaits its next direction.
* **If** it **runs across** a naval **mine (**'**\***'**)**, the submarine takes serious **damage**. When a mine is blown, the position of the mine will be marked with '**-**' (dash). U-9 can **withstand two hits** from naval mines. The **third time** the submarine **is hit** by a mine, it **disappears** and the **mission** **is** **failed. The battle is over** and the following message should be printed on the Console: "**Mission failed, U-9 disappeared! Last known coordinates [{row}, {col}]!**"
* If a battle cruiser is reached **(**'**C**'**)**, the submarine destroys it and the position of the destroyed cruiser will be marked with '**-**' (dash).
* If this is the **last** (third) battle **cruiser** on the **battlefield**, **the battle is over** and the following message should be printed on the Console: "**Mission accomplished, U-9 has destroyed all battle cruisers of the enemy!**"

**The program will end when the battle is over (All battle cruisers are destroyed or the submarine hits mines three times).**

### Input

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

### Output

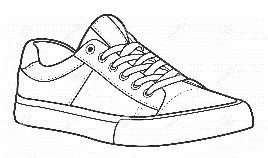
* If all battle **cruisers** are destroyed, print: "**Mission accomplished, U-9 has destroyed all battle cruisers of the enemy!**"
* If U-9 is hit by a mine three times, print: "**Mission failed, U-9 disappeared! Last known coordinates [{row}, {col}]!**".
* At the end, print the **final state** of the matrix (**battlefield**) with the **last known U-9’s position** on it.

### Constraints

* The size of the **square** matrix (**battlefield**) will be between **[4…10].**
* U-9’s starting position will always be marked with '**S**'.
* There will be always three battle cruisers - fields marked with '**C**'.
* There will be always enough mines on the battlefield to destroy the submarine.
* The commands given will direct the submarine only within the limits of the battlefield.
* You will always receive enough commands to end the battle.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  \*--\*-  -S-\*C  -\*---  -----  -C-\*C  right  down  left  up  right  right  right  down  down  down  up  left  left  left  down | Mission accomplished, U-9 has destroyed all battle cruisers of the enemy!  \*--\*-  -----  -----  -----  -S-\*- |
| 5  \*--\*-  -S-\*C  -\*---  -----  \*C-\*C  right  right  up  left  left  left | Mission failed, U-9 disappeared! Last known coordinates [0, 0]!  S----  ----C  -\*---  -----  \*C-\*C |

****

**Shoe Store**

You can test your solutions in [Judge](https://judge.softuni.org/Contests/Practice/Index/3625#2)

1. **Preparation**

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

1. **Problem description**

*Write a program that keeps track of the shoes in a shoe store.*

**Shoe**

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

* **Brand – string**
* **Type – string**
* **Size – double**
* **Material – string**

The class **constructor** should receive **brand, type, size and material**.

Override **ToString()** method: "Size {Size}, {Material} {Brand} {Type} shoe."

**ShoeStore**

Next, you are given a class named **ShoeStore,** which has a **collection**of type **Shoe**. The name of the collection should be **Shoes, which could not be modified directly**. All the entities of the shoe collection have the **same** properties. The **ShoeStore** also should have the following properties:

* **Name – string**
* **StorageCapacity – int**
* **Shoes – List<Shoe>**

The **constructor** of the **ShoeStore** class should receive **name and storageCapacity**. It should initialize also the **Shoes** with a new instance of the collection.

Implement the following features:

* Getter **Count** - returns the **total count** of the shoes in the storage room.
* **string AddShoe(Shoe shoe)** – **adds** a **Shoe** to the **Shoes** collection and returns: **"**Successfully added {shoeType} {shoeMaterial} pair of shoes to the store.**"** 
  + If the **StorageCapacity** doesn’t allow adding more shoes in the **Store**

(Shoes.Count == StorageCapacity), returns: **"**No more space in the storage room.**"**

* **int RemoveShoes(string material)** – removes all shoes by a **given material,** as a result, return the **count** of the shoes which were removed**.**
* **List<Shoe> GetShoesByType(string type)** – searches and returns a **list of shoes** by **given type. Search should be case insensitive.**
* **Shoe GetShoeBySize(double size)** – return the first shoe, with the given size**.**
* **string StockList(double size, string type)** –returns a string with information about the shoes which match the given size and typein the following format:
  + If **there are** any **pairs mathcing** the given parameters, print the following report on the Console:

**"Stock list for size {size} - {type} shoes:  
{Shoe1}  
{Shoe2}  
{…}**"

* + If **none of the pairs match** the given parameters, print the following message on the console:

**"No matches found!**"

**Note: Do not use** "\n\r" **for a new line.**

**Constraints**

* You will always have a pair of shoes added before receiving methods that manipulate the **shoes in the ShoeStore**.

**Examples**

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

|  |
| --- |
| **Sample code usage** |
| var store = new ShoeStore("SportiveNation", 10);  var shoeOne = new Shoe("Nike", "running", 42.5, "textile");  var shoeTwo = new Shoe("Salomon", "hiking", 42, "textile");  var shoeThree = new Shoe("Reebok", "running", 38, "textile");  var shoeFour = new Shoe("LaCoste", "casual", 40.5, "leather");  var shoeFive = new Shoe("Adidas", "casual", 39, "textile");  var shoeSix = new Shoe("Nike", "hiking", 42.5, "textile");  var shoeSeven = new Shoe("Adidas", "casual", 42, "leather");  var shoeEight = new Shoe("AirJordan", "running", 42, "leather");  var shoeNine = new Shoe("CalninKlein", "casual", 41.5, "leather");  var shoeTen = new Shoe("Puma", "hiking", 42, "textile");  var shoeEleven = new Shoe("Skechers", "casual", 42.5, "leather");  Console.WriteLine(store.AddShoe(shoeOne));  // Successfully added running textile pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeTwo));  // Successfully added hiking textile pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeThree));  // Successfully added running textile pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeFour));  // Successfully added casual leather pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeFive));  // Successfully added casual textile pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeSix));  // Successfully added hiking textile pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeSeven));  // Successfully added casual leather pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeEight));  // Successfully added running leather pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeNine));  // Successfully added casual leather pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeTen));  // Successfully added hiking textile pair of shoes to the store.  Console.WriteLine(store.AddShoe(shoeEleven));  // No more space in the storage room.  var runningShoes = store.GetShoesByType("Running");  var hikingShoes = store.GetShoesByType("hIKING");  Console.WriteLine(string.Join($"{Environment.NewLine}", runningShoes));  // Size 42.5, textile Nike running shoe.  // Size 38, textile Reebok running shoe.  // Size 42, leather AirJordan running shoe.  Console.WriteLine(string.Join($"{Environment.NewLine}", hikingShoes));  // Size 42, textile Salomon hiking shoe.  // Size 42.5, textile Nike hiking shoe.  // Size 42, textile Puma hiking shoe.  Console.WriteLine(store.RemoveShoes("leather"));  // 4  var shoeBySize = store.GetShoeBySize(42.5);  Console.WriteLine(shoeBySize);  // Size 42.5, textile Nike running shoe.  Console.WriteLine(store.StockList(42, "hiking"));  //Stock list for size 42 - hiking shoes:  //Size 42, textile Salomon hiking shoe.  //Size 42, textile Puma hiking shoe. |

**Submission**

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