# Exam Preparation

Part I: [judge.softuni.bg/Contests/691/Programming-Fundamentals-Exam-09-July-2017-Part-1](https://judge.softuni.bg/Contests/691/Programming-Fundamentals-Exam-09-July-2017-Part-1)  
Part II: [judge.softuni.bg/Contests/692/Programming-Fundamentals-Exam-09-July-2017-Part-2](https://judge.softuni.bg/Contests/692/Programming-Fundamentals-Exam-09-July-2017-Part-2)

# Problem 1. Poke Mon

A Poke Mon is a special type of pokemon which likes to Poke others. But at the end of the day, the Poke Mon wants to keeps statistics, about how many pokes it has managed to make.

The Poke Mon pokes his target, and then proceeds to poke another target. The **distance** between his **targets** **reduces** his **poke power**.

You will be **given** the **poke power** the Poke Mon has, **N** – an **integer**.

Then you will be **given** the **distance** between the **poke targets**, **M** – an **integer**.

Then you will be **given** the **exhaustionFactor** **Y** – an **integer**.

Your task is to start **subtracting** **M** from **N** until **N** becomes **less than M**, i.e. the Poke Mon does not have enough power to reach the next target.   
**Every time** you **subtract M** from **N** that means you’ve reached a **target** and poked it successfully. **COUNT** how **many targets** you’ve poked – **you’ll need** that **count**.

The Poke Mon becomes gradually more exhausted. **IF** **N** **becomes equal** to **EXACTLY 50 %** of its **original value**, you must **divide** **N** by **Y**, if it is **POSSIBLE**. **This** **DIVISION** is between **integers**.

If a division is **not possible**, you should **NOT** do it. Instead, you should continue **subtracting**.

**After dividing**, you should **continue** subtracting from **N**, until it becomes **less** than **M**.

When **N** becomes **less** than **M**, you must take **what has remained** of **N** and the **count** of **targets** you’ve poked, and print them as output.

**NOTE**: When you are **calculating percentages**, you should be **PRECISE** at **maximum**.

**Example**: 505 is **NOT EXACTLY 50 %** from 1000, its **50.5 %**.

### Input

* The input consists of **3 lines**.
* On the **first line** you will receive **N** – an **integer**.
* On the **second line** you will receive **M** – an **integer**.
* On the **third line** you will receive **Y** – an **integer**.

### Output

* The output consists of **2 lines**.
* On the **first line** print **what has remained** of **N**, after **subtracting** from it.
* On the **second line** print the **count** of **targets**, you’ve managed to poke.

### Constrains

* The integer **N** will be in the **range** **[1, 2.000.000.000]**.
* The integer **M** will be in the **range** **[1, 1.000.000]**.
* The integer **Y** will be in the **range** **[0, 9]**.
* Allowed time / memory: **16 MB / 100ms**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  2  3 | 1  2 | N = 5, M = 2, Y = 3.  We start **subtracting** **M** from **N**.  **N – M = 3**. **1** target poked.  **N – M = 1**. **2** targets poked.  **N < M**. We print **what has remained** of **N**, which is **1**.  We print the **count of targets**, which is 2. |
| 10  5  2 | 2  1 | N = 10, M = 5, Y = 2.  We start **subtracting** **M** from **N**.  **N** **–** **M** = **5**. (N is still not less than M, they are equal).  **N** became **EXACTLY** **50 %** of its **original value**.  **5** is **50 %** from **10**. So we divide **N** by **Y**.  **N / Y** = **5 / 2** = **2**. (**INTEGER DIVISION**). |

# Problem 2. Pokemon Don’t Go

Ely likes to play Pokemon Go a lot. But Pokemon Go bankrupted … So the developers made Pokemon Don’t Go out of depression. And so Ely now plays Pokemon Don’t Go. In Pokemon Don’t Go, when you walk to a certain pokemon, those closer to you, naturally get further, and those further from you, get closer.

You will receive a **sequence** of **integers**, separated by **spaces** – the distances to the pokemons.  
Then you will begin **receiving integers**, which will **correspond** to **indexes** in **that** **sequence**.

When you **receive** an **index**, you must **remove** the **element** at **that index** from the **sequence** (as if you’ve captured the pokemon).

* You must **INCREASE** the **value** of **all elements** in the sequence which are **LESS** or **EQUAL** to the **removed element**, with the **value** of the **removed element**.
* You must **DECREASE** the **value** of **all elements** in the sequence which are **GREATER** than the **removed element**, with the **value** of the **removed element**.

If the **given index** is **LESS** than **0**, **remove** the **first element** of the **sequence**, and **COPY** the **last element** to its place.

If the **given index** is **GREATER** than the **last index** of the **sequence**, **remove** the **last element** from the sequence, and **COPY** the **first element** to its place.

The **increasing** and **decreasing** of elements should be done in these cases, **also**. The **element**, whose value you should use is the **REMOVED** element.

The program **ends** when the **sequence** has **no elements** (there are no pokemons left for Ely to catch).

### Input

* On the **first line** of input you will receive a **sequence** of **integers**, **separated** by **spaces**.
* On the **next several** lines you will receive **integers** – the **indexes**.

### Output

* When the program ends, you must print on the console, the **summed up** **value** of **all REMOVED elements**.

### Constrains

* The input data will consist **ONLY** of **valid integers** in the **range [-2.147.483.648, 2.147.483.647]**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 4 5 3  1  1  0 | 14 | The **array** is {4, 5, 3}. The index is 1.  We **remove** 5, and we **increase all** **lower** than it and **decrease all higher** than it.  In this case there are **no higher** than 5.  The result is {9, 8}.  The **index** is 1. So we remove 8, and **decrease all higher** than it.  The result is {1}.  The **index** is 0. So we remove 1.  There are **no elements** **left**, so we print the **sum** of **all removed elements**.  5 + 8 + 1 = 14. |
| 5 10 6 3 5  2  4  1  1  3  0  0 | 51 | **Step 1**: {11, 4, 9, 11}  **Step 2**: {22, 15, 20, 22}  **Step 3**: {7, 5, 7}  **Step 4**: {2, 2}  **Step 5**: {4, 4}  **Step 6**: {8}  **Step 7**: {} **(empty).**  **Result** = 6 + 11 + 15 + 5 + 2 + 4 + 8 = 51. |

## maxresdefault Problem 3. Regexmon

Bojomon and Didimon are both pokemons which like to capture specific elements from a text. But they always fight so they decided to take turns in capturing things. Didimon is **first**, of course, because “ladies first”.

**Bojomon** matches only **pokewords**.   
A **pokeword** consists of **2 sequences** of **alphabet letters**, separated by a **dash** (‘-’). **Example**: “pika-pika”

**Didimon** matches everything **EXCEPT** **alphabet letters** and **dashes**.

You will get a **single string**, containing random ASCII characters, from the input.

**Didimon starts first**. If she **finds** a **match**, she prints it on the console, and **switches** **turns** with Bojomon.

Then Bojomon starts matching **AFTER** **Didimon’s match**, **IGNORING** everything **before** it, from the string. When he finds a match, he prints it and **switches turns** with **Didimon**, and she starts matching **AFTER** Bojomon’s match.

This process **repeats** until **the one** who is **at turn** does **NOT** match **anything** till the end of the string.

**Example**: “!!!aa-aa?\_?”

1. **Didimon** matches “!!!”, then she **switches** with Bojomon.
2. Bojomon **starts after Didimon’s match** and so he has “aa-aa?\_?”. He matches “aa-aa” and **switches** with Didimon.
3. Didimon **starts after Bojomon’s match** and so she has “?\_?”. She matches “?\_?” and **switches** with Bojomon again.
4. Bojomon **starts after Didimon’s match** and so he has “” (**empty string**). He matches **nothing** and the program **ends**.

### Input

* The input will consist of a **single line**, containing the **text**, that Bojomon and Didimon are going to match.

### Output

* As output you should print Bojomon and Didimon’s **matches**, each on a **new line**.

### Constrains

* The input string may contain **any ASCII character**.
* Allowed time / memory: **100ms / 16MB**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ^^^^pika-pika^^^^ | ^^^^  pika-pika  ^^^^ |
| !.!asd-asd---??!chuuuu!!!asd-dsa | !.!  asd-asd  ??!  asd-dsa |

# Problem 4. Pokemon Evolution

You have been tasked to keep track of pokemons and their evolutions. A pokemon can evolve in several phases and types. When it evolves, the pokemon has an evolution index, which indicates how much it has evolved.

You will receive input lines in the following format:  
{pokemonName} -> {evolutionType} -> {evolutionIndex}

The pokemonName and evolutionType will be **strings**. The evolutionIndex will be an **integer**. Your task is to store every **pokemon** and his **evolutions**.   
If you receive an existent pokemonName, you should **add** the **new** **evolution** to it.

A single **pokemon** may have **many evolutions** with the **same type** and the **same index**.

In some rare cases you may receive the following input:  
{pokemonName}

When you receive only a pokemonName, you must **check if there is** such a **pokemon**, and if there is, you must print all of its **evolutions** by **order of input**.

The **input sequence ends** when you **receive** the command “wubbalubbadubdub”.   
Then you must print all pokemons and their evolutions. The pokemons must be printed by **order of input**. Each **pokemon’s evolutions** must be **ordered** by **evolution index** in **descending order**.

### Input

* The input will come in the form of lines in the format specified above.
* In some rare cases you may have only one element of the input – the pokemonName.
* The input sequence ends when you receive the command “wubbalubbadubdub”.

### Output

* **Pokemons** and their **evolutions** must be printed in the following format:

“# {pokemoName}  
 {evolution1Type} <-> {evolution1Index}  
 {evolution2Type} <-> {evolution2Index}

…”

* If you have received a pokemonName and you are **printing its evolutions**, the order is – by **order of input**.
* If you have received the **ending command**, and you are printing the **pokemons’ evolutions**, the order is – by evolutionIndex in **descending order**.

### Constrains

* The pokemonName and evolutionType are strings which may contain any ASCII character   
  (except ‘-’, ‘ ’, ‘>’).
* The evolutionIndex will be an **integer** in **range [0, 1.000.000.000]**.
* There will be **NO invalid** input data.
* Allowed time / memory: **100ms / 16 MB**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Ekans -> Hybrid -> 100  Nidoran -> Physical -> 150  Ekans -> Psychological -> 50  Jigglypuff -> Hybrid -> 1000  Jigglypuff -> Physical -> 2000  wubbalubbadubdub | # Ekans  Hybrid <-> 100  Psychological <-> 50  # Nidoran  Physical <-> 150  # Jigglypuff  Physical <-> 2000  Hybrid <-> 1000 |
| Pikachu -> Hybrid -> 100  Meowth -> Physical -> 100  Pikachu -> Psychological -> 50  Meowth -> Physical -> 50  Pikachu -> Hybrid -> 150  Meowth  Pikachu  wubbalubbadubdub | # Meowth  Physical <-> 100  Physical <-> 50  # Pikachu  Hybrid <-> 100  Psychological <-> 50  Hybrid <-> 150  # Pikachu  Hybrid <-> 150  Hybrid <-> 100  Psychological <-> 50  # Meowth  Physical <-> 100  Physical <-> 50 |

## b56f3c0f767242e9a52b947a2b80436877d733b0_hq