## Motion Bootcamp – Entry Exam – 23.02.2020

## Materials

Write a program that calculates the **total amount** of a material that can be extracted from a source. The source has a **starting yield N** (your input), which indicates how much material can be mined on the **first day**. After it has been mined for a day, the **yield drops** by 10, meaning on the second day it’ll produce 10 less material than on the first, on the third day 10 less than on the second, and so on (see examples). A source is considered profitable only while its yield is **at least** 100 (>= 100) – when less than 100 material is expected in a day, abandon the source.

The mining crew **consumes** 26 materials **every day** at the end of their shift and an **additional** 26 after the mining is over. Note that the workers cannot consume more materials than they have mined so far.

When the operation is complete, print on the console on two separate lines how many days the mine has operated and the total amount of material left (subtract the consumed from the extracted).

### Input

**N** –the starting yield for day 1 - an integer within the range of [0…2^28]

### Output

Print on the **console** on two separate lines how many **days** the mine has operated and the **total amount** of material left (subtract the consumed from the extracted)

Note: **For JavaScript solutions the input passed as a parameter!**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Explanation** |
| 111 | 2  134 | **Day 1** we extract **111** material and at the end of the shift, the workers consume **26**, leaving  **85**. The yield drops by **10** to **101**.  **Day 2** we extract **101** material, the workers consume **26**, leaving **75**. Adding that to the material from the previous day gives us **160** and the yield drops to **91**.  Since the expected yield is less than **100**, we abandon the source. The workers take another **26**, leaving a total of **134**. The mine has operated for **2** days. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 450 | 36  8938 | 200 | 11  1338 |

## Generator

Write a program, which gets **two** **numbers** (**N** and **K**) and generates 5-letter strings **ordered** **alphabetically,** which represent **all possible** combinations consisting of the following five symbols:

* **Symbol 1**: digit from 1 to **N**
* **Symbol 2**: digit from 1 to **N**
* **Symbol 3**: letter (a-z), which is amongst the first **K** letters of the alphabet
* **Symbol 4**: letter (a-z), which is amongst the first **K** letters of the alphabet
* **Symbol 5**: digit from 1 to **n**, **larger** than the first two digits

### Input

The input consists of two numbers (**N** and **K**) read from the console, or **passed as parameters to the function(JS)**.

### Output

You should print all combinations, **ordered** **alphabetically**, separated by **space**

### Examples

|  |  |
| --- | --- |
| **вход** | **изход** |
| 2  4 | 11aa2 11ab2 11ac2 11ad2 11ba2 11bb2 11bc2 11bd2 11ca2 11cb2 11cc2 11cd2 11da2 11db2 11dc2 11dd2 |
| 3  1 | 11aa2 11aa3 12aa3 21aa3 22aa3 |
| 3  2 | 11aa2 11aa3 11ab2 11ab3 11ba2 11ba3 11bb2 11bb3 12aa3 12ab3 12ba3 12bb3 21aa3 21ab3 21ba3 21bb3 22aa3 22ab3 22ba3 22bb3 |
| 4  2 | 11aa2 11aa3 11aa4 11ab2 11ab3 11ab4 11ba2 11ba3 11ba4 11bb2 11bb3 11bb4 12aa3 12aa4 12ab3 12ab4 12ba3 12ba4 12bb3 12bb4 13aa4 13ab4 13ba4 13bb4 21aa3 21aa4 21ab3 21ab4 21ba3 21ba4 21bb3 21bb4 22aa3 22aa4 22ab3 22ab4 22ba3 22ba4 22bb3 22bb4 23aa4 23ab4 23ba4 23bb4 31aa4 31ab4 31ba4 31bb4 32aa4 32ab4 32ba4 32bb4 33aa4 33ab4 33ba4 33bb4 |

## Train

You will receive input lines in one of the following formats:

* **{townName}:{time}->{passengersCount}**
* **{townName}:ambush->{passengersCount}**
* **Slide rule**

**{townName}:{time}->{passengersCount}**

If you receive the line above, the train has travelled to **certain town** for a **certain amount of time** with **certain count of passengers**. You need to keep track **for each town**. You have to memorize **the fastest time that any train has reached** a town and the **total count (sum) of passengers for each town.**

**{townName}:ambush->{passengersCount}**

If you receive the line above, somewhere along the track to the current town, the train was **ambushed** and the passengers **can't reach** their destination**.** If this happens you need to **set the time record** for this town to **"0"** andyou should **remove** the current **count** of passengers from the total count**.** If it's the **first time this train travels to this town** then you **simply ignore this line.**

**Slide rule** you end the program and print data for each town in the following format:

**{townName} -> Time: {fastestTime} -> Passengers: {totalCountPassengers}**

The output should be ordered by best time and then by town's name. If a town is with **no recorded time** (the time is equal to 0)or there are **no passengers** (count is equal or less than 0) you **should not print** it.

### Input / Constraints

* Until you receive **Slide rule** you will be receiving participant submissions in one of the formats specified above
* The time will always be **positive** **integer** in the range **[1-1000]**
* The count of passengers will always be **positive integer** in the range **[1-100000]**

### Output

* Print recorded data in the following format:  
  **{townName} -> Time: {bestTime} -> Passengers: {totalCountPassengers}**The output should be ordered by best time and then by town's name. If a town is with **no recorded time** (the time is equal to 0)or there are **no passengers** (count is equal or less than 0) you **should not print** it.
* Allowed working time / memory: **100ms** / **16MB**

Note: **For JavaScript solutions the input will be an array of strings, passed as a parameter!**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| Sto-Lat:8->120  Ankh-Morpork:3->143  Sto-Lat:9->80  Ankh-Morpork:4->143  Sto-Lat:3->20  Quirm:12->40  Quirm:13->29  Slide rule | Ankh-Morpork -> Time: 3 -> Passengers: 286  Sto-Lat -> Time: 3 -> Passengers: 220  Quirm -> Time: 12 -> Passengers: 69 | We have Sto-Lat multiple times, but we keep only the best time equal to 3 with the total count of passengers equal to 220. Ankh- Morpork is with fastest time 3, so we compare those two by names. Quirm comes third with time of 12. |
| Quirm:12->258  Ankh-Morpork:ambush->200  Ankh-Morpork:3->143  Sto-Lat:4->80  Ankh-Morpork:4->143  Ankh-Morpork:ambush->143  Sto-Lat:3->20  Ankh-Morpork:5->17  Slide rule | Sto-Lat -> Time: 3 -> Passengers: 100  Ankh-Morpork -> Time: 5 -> Passengers: 160  Quirm -> Time: 12 -> Passengers: 258 | The record time for Ankh-Morpork is equal to 5 since the previous one was set to 0 during the ambush. Note that we keep the count of passengers. |

## Swaps

Write a program which gets a matrix as an input and performs certain operations with its elements.

Your program should then receive commands in format: **swap row1 col1 row2 col2**  
Where **row1, row2, col1, col2** are **coordinates** in the matrix. In order for a command to be valid, it should start with the **swap** keyword along with **exactly** **four** valid coordinates. You should **swap** the values at the given coordinates (cell [row1, col1] with cell [row2, col2]) and **print** the matrix at **each step** (thus you'll be able to check if the operation was performed correctly).

If the **command is not valid** (doesn't contain the keyword **swap**, has fewer or more coordinates entered or the given coordinates **do not exist**), print **Invalid input!** and move on to the next command. Your program should finish when the string **END** is entered.

### Input / Constraints

* Line 1:  
  **R C**  
  two integers in the range [1-1000], representing respectively the number of rows and the number of columns of the matrix
* Lines 2 – **R**+1:  
  **C** number of **strings**, representing a row in the matrix
* Lines **R** + 2 and beyond:  
  commands in the above-described format
* **END**keyword for end of input

### Output

* Each valid command input should print the matrix (**R** rows per operation)
* Each invalid command should output **Invalid input!**

Note: **For JavaScript solutions the input will be an array of strings, passed as a parameter!**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 3  1 2 3  4 5 6  swap 0 0 1 1  swap 10 9 8 7  swap 0 1 1 0  END | 5 2 3  4 1 6  Invalid input!  5 4 3  2 1 6 |
| 1 2  Hello World  0 0 0 1  swap 0 0 0 1  swap 0 1 0 0  END | Invalid input!  World Hello  Hello World |