# Problem 1. SoftUni Coffee Orders

We are placing **N** orders at a time. You need to calculate the price after the discount based on the following formula:

((daysInMonth \* capsulesCount) \* pricePerCapsule)

\***Hint** – The DateTime class may come in handy to calculate the days in month.

### Input / Constraints

* On the first line you will receive integer **N** – the count of orders the shop will receive.
* For each order you will receive the following information:
  + Price per capsule - **floating-point number in range [0…79,228,162,514,264,337,593,543,950,335]**.
  + Order date - in the following format {d/M/yyyy}, e.g. 25/11/2016, 7/03/2016, 1/1/2020.
  + Capsules count - **integer in range [0…2,147,483,647]**.

The input will be in the described format, there is no need to check it explicitly.

### Output

The output should consist of **N + 1** lines. For each order you must print a single line in the following format:

* “The price for the coffee is: ${**price**}”

On the last line you need to print the total price in the following format:

* “Total: ${totalP**rice**}”

The **price must be rounded** to 2 decimal places.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1  1.53  06/06/2016  8 | The price for the coffee is: $367.20  Total: $367.20 | We are given only 1 order. Then we use the formulas:  **orderPrice** = 30 (days in June 2016) \* 8 \* 1.53 = 367.20 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** |  |
| 2  4.99  6/07/2016  3  0.35  03/01/2013  5 | The price for the coffee is: $464.07  The price for the coffee is: $54.25  Total: $518.32 |

# Problem 2 – Command Interpreter

Jagged arrays, regular expressions, asynchronous programming… Tough stuff. But simple structures like arrays are piece of cake, right? Let’s see how well you can manipulate data in a collection.

You will be given a series of strings on a single line, separated by one or more whitespaces. These represent the collection you’ll be working with.

On the next input lines, until you receive the command **"end"**, you’ll receive a series of commands in one of the following formats:

* **"reverse from [start] count [count]"** – this instructs you to reverse a **portion** of the array – just [count] elements starting at index [start];
* **"sort from [start] count [count]"** – this instructs you to sort a **portion** of the array - [count] elements starting at index [start];
* **"rollLeft [count] times"** – this instructs you to move **all** elements in the array to the left [count] times. On each roll, the first element is placed at the end of the array;
* **"rollRight [count] times"** – this instructs you to move **all** elements in the array to the right [count] times. On each roll, the last element is placed at the beginning of the array;

If any of the provided indices or counts is **invalid** (non-existent or negative), you should print a message on the console – **"Invalid input parameters."** and **keep the collection unchanged.**

After you’re done, print the resulting array in the following format: **"[arr0, arr1 … arrN]"**. The examples should help you understand the task better.

### Input

* The input data should be read from the console.
* The first input line will hold **a series of strings**, separated by **one or more whitespaces**.
* The next lines will hold **commands** in the described formats (exactly).
* The input ends with the keyword **"end".**
* The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output should be printed on the console.
* Each time an invalid command is received (containing an invalid index or count parameter), print the following line: **"Invalid input parameters."**
* After receiving the "**end**" command, print the **resulting array** on the console in the format specified above.

### Constraints

* The **count of strings** in the collection will be in the range [1 … 50].
* In the **sorting** of the **strings**, the **lowercase** letters come **BEFORE** the **uppercase**.
* The **number of commands** will be in the range [1 … 20].
* All commands will be in the described format; an invalid command is a command containing invalid [start] or [count], **there won’t be any missing or misspelled words**.
* [**start**] and [**count**] will be integers in the range [-231 … 231 - 1].
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 **5 8 7 3** 10 6 4 9  reverse from 2 count 4  end | [1, 2, **3, 7, 8, 5**, 10, 6, 4, 9] |

# Problem 3 – Rage Quit

Every gamer knows what rage-quitting means. It’s basically when you’re just not good enough and you blame everybody else for losing a game. You press the CAPS LOCK key on the keyboard and flood the chat with gibberish to show your frustration.

Chochko is a gamer, and a bad one at that. He asks for your help; he wants to be the most annoying kid in his team, so when he rage-quits he wants something truly spectacular. He’ll give you **a series of strings followed by non-negative numbers**, e.g. "a3"; you need to print on the console **each string repeated N times**; **convert the letters to uppercase beforehand**. In the example, you need to write back "AAA".

On the output, print first a statistic of the **number of unique symbols** used (the casing of letters is irrelevant, meaning that 'a' and 'A' are the same); the format shoud be **"Unique symbols used {0}"**. Then, **print the rage message** itself.

The **strings and numbers will not be separated by anything**. The input will always start with a string and for each string there will be a corresponding number. The entire input will be given on a **single line**; Chochko is too lazy to make your job easier.

### Input

* The input data should be read from the console.
* It consists of a single line holding a series of **string-number sequences**.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output should be printed on the console. It should consist of **exactly two lines**.
* On the first line, print the **number of unique symbols used** in the message.
* On the second line, print the **resulting rage message** itself.

### Constraints

* The count of **string-number pairs** will be in the range [1 … 20 000].
* Each string will contain any character **except digits**. The **length** of each string will be in the range [1 … 20].
* The **repeat count** for each string will be an integer in the range [0 … 20].
* Allowed working time for your program: 0.3 seconds. Allowed memory: 64 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| a3 | Unique symbols used: 1  AAA | We have just one string-number pair. The symbol is 'a', convert it to uppercase and repeat 3 times: AAA.  Only one symbol is used ('A'). |
| aSd2&5s@1 | Unique symbols used: 5  ASDASD&&&&&S@ | "aSd" is converted to "ASD" and repeated twice; "&" is repeated 5 times; "s@" is converted to "S@" and repeated once.  5 symbols are used: 'A', 'S', 'D', '&' and '@'. |

'

# Problem 4. Files

You are given number of files with their **full file paths** and **file sizes**. You need to **print all file names** with **a given extension** that are present in **a given root directory sorted by their file size** in **descending order**. If two files have **same size**, order them **by alphabetical order**.

If a **file name (file name + extension) appears more than once in a given root**, **save only its latest value.** If a file name appears in more than one root, they are treated as different files.

If there **aren't any files** that correspond to the query, print **"No"**.

## Input / Constrains

* On the **first line** of input you will get **N** the **number of files to be read from the console**
* On the next N lines, you receive **the actual files in the format "root\folder\filename.extension;filesize"**
* There may be more than one folder e.g. **files can be deeply nested**
* On the last line you receive a query string in format **"{extension} in {root}"**. You need to print all files with the given extension that are in present in the given root

## Output

* You need to print all files sorted by their size in descending order.
* If two files have the same size, order them by alphabetical order.
* In the **sorting** of the **strings**, the **lowercase letters** come **BEFORE** the **uppercase**.
* Files should be printed in the given format **"filename.extension - filesize KB"**
* If there **aren't any movies** that correspond to the query, print **"No"**.

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Windows\Temp\win.exe;5423  Games\Wow\wow.exe;1024  Games\Wow\patcher.cs;65212  Games\Pirates\Start\keygen.exe;1024  exe in Games | keygen.exe - 1024 KB  wow.exe - 1024 KB |
| 3  C:\Documents\01. problems.docx;6521  D:\Documents\02. Documents\ presentation.pptx;44234  E:\Movies\Classics\someclassicmovie.avi;6221235212  docx in E: | No |