Python Fundamentals Exam @ Soft Uni 2017

September 3, 2017

1 Boza queue

Description

In the all new company Boza Soft they've opened a new kind of fun for programmers - Boza bar. This is a place where all the programmers gather and wait for a EXTREMLY LARGE GLASS OF THE GREAT BOZA. But as you know the programmers are quite strange creatures and in this queue for boza strange things happen - the queue executes instructions from the given instruction set:

- **"ADD *Name*"** - adds new programmer into the queue
- **"NEXT"** - gives the current programmer boza (prints its name) and removes it from the queue
- **"SKIP"** - removes the current programmer from the queue
- **"REM *Name*"** - removes all the programmers with this name from the queue
- **"CLEANUP"** - gives boza to all the programmers with even name length and removes them from the queue
- **"CLEANDOWN"** - removes all the programmers with even name length and odd index (counting from zero) from the queue

Input

- On the input there will be only valid commands
- The program accepts commands until the queue gets empty

Output

The names of the programmers in the order they got boza separated by a new line

Constrains

- The first command is always **"ADD"**
- Allowed working time: 1 sec.
- Allowed memory consumption: 16 MB

Examples

Input	Output
ADD PESHO ADD GOSHO NEXT ADD PENKA NEXT ADD PLOKIJUH NEXT SKIP	PESHO GOSHO PENKA
ADD PESHO ADD IVAN ADD ASDDSAASD ADD ASDF CLEANUP NEXT NEXT	IVAN ASDF PESHO ASDDSAASD
ADD PESHO ADD IVAN ADD ASDF ADD ASDDSAASD CLEANDOWN REM ASDDSAASD NEXT NEXT	PESHO ASDF

2 Alex & Dimo at SoftUniniada 2017

Description

All matches with real people and events are not accidental One day Alex and Dimo had to present their project at SoftUni as part of the contest "SoftUniada 2017". So they decided to check if the demo was working just before their presentation and Alex found that the database was not working. While Dimo was panicking because some bad people were disassembling a Spitfire Mk IX to recycle it, Alex replaced the database with something magical. The database contained information for result of all users in all contest divided by tasks.
Prompted by this case we decide to give you the following task: You must simulate a ranking system (judge system). What is that? You must implement a program that receives a lot of commands as following and answers a lot of queries:

- ADD SOLUTION %USER% %CONTEST% %TASK% %SOLUTION_ID% %POINTS% → initializes new solution if solution %SOLUTION_ID% doesn't exist or print "Solution already exists" otherwise
- GET SOLUTION %SOLUTION_ID% → prints %USER%, %CONTEST%, %TASK% and %POINTS% from the solution information if the solution exists or prints "Solution not found" otherwise.
- ADD CERTIFICATE %USER% %CONTEST% → initializes a certificate for %USER% from %CONTEST% if he/she doesn't already have or prints "Certificate already exists" otherwise.
- GET CERTIFICATE %USER% %CONTEST% → prints total sum of maximum given points of all task in %CONTEST% if the certificate is initialized or print "Certificate not found" otherwise
- GET CERTIFICATES %USER% → prints list of all contests completed with a certificate by %USER% sorted by contest name (increasing)
- GET RANKLIST %CONTEST% → prints list of all participants sorted by total score (decreasing)
  Output format:
  USER1,USER2,USER3,USER4...
- QUIT → means end of input

Input

Input is a sequence of commands which ends with "QUIT" command

Output

Output is a set of output data of each command ordered as they are in input

Constrains

- %USER%, %CONTEST%, %SOLUTION_ID% and %TASK% will be texts which don't contains spaces (' ')
- %POINTS% will be floating-point number in range [0, 100]
- Allowed working time: 1 sec.
- Allowed memory consumption: 256 MB

Examples

Input	Output
ADD SOLUTION TTS PythonFundamentals 2 100 50 ADD SOLUTION TTS PythonFundamentals 2 100 100 ADD SOLUTION TTS PythonFundamentals 2 101 100 GET SOLUTION 100 GET SOLUTION 101 GET SOLUTION 102 ADD CERTIFICATE TTS PythonFundamentals GET CERTIFICATE TTS PythonFundamentals GET CERTIFICATE TTS PythonBasics ADD SOLUTION TTS PythonBasics 1 102 50 ADD CERTIFICATE TTS PythonBasics GET CERTIFICATES TTS ADD SOLUTION Ro6afF PythonFundamentals 2 103 100 ADD SOLUTION Ro6afF PythonFundamentals 3 104 100 GET RANKLIST PythonFundamentals QUIT	Solution already exists User: TTS, Contest: PythonFundamentals, Task: 2, Points: 50.0 User: TTS, Contest: PythonFundamentals, Task: 2, Points: 100.0 Solution not found 100.0 Certificate not found PythonBasics, PythonFundamentals Ro6afF, TTS

3 Pesho's secret conversation

Description

Pesho loves to chat with Penka but his parents really like to follow his conversations. One day Pesho invented revolution way to code the messages:

- First he generate decrypting key which is a random permutation of all Latin letters (either small and capital)
  This permutation presents each letter transformation where the index of the letter is the number of original letter in the alphabet and the value (the letter in permutation) is its transformation
  The indexes are:
  - a-z ⇒ 0-25
  - A-Z ⇒ 26-51
- After that he transforms each letter of his original message as it said in the key.
- After that Pesho transforms each letter of the encrypted message to an integer which is its index
- After that he sends the key and the encrypted message concatenated

Input

- The first line of the standard input is encrypted message from Penka.
- The second line contains Pesho's key and his original message

Output

- The first line of standard output must be decrypted Penka's message
- The second line must be encrypted Pesho's message

Constrains

- Pesho's original message can contain symbols different than letters and you must ignore them (including spaces - ' ')
- Allowed working time: 1 sec.
- Allowed memory consumption: 16 MB

Examples - Download the test

Line	Input	Output
1	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ00010203040506070809101112131415161718192021222324252627282930313233343536373839404142434445464748495051	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
2	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ abcd.e.f.g,h#i4j5klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ00010203040506070809101112131415161718192021222324252627282930313233343536373839404142434445464748495051

4 PLOKIJUH ordering

Description

In the BalkanBozaHard they've made a brand new tool for roasting PLOKs - the very PLOKIJUH! And they are making a web application to sell them. But there are a lot of different types of PLOKIJUHs. So we have to make an API program that handles the collection of PLOKIJUHs in their company (don't worry, the program works with the standard input and output - the console). Every PLOKIJUH has the following properties:

- Name - a string
- Price - a floating-point number
- A bunch of bozatronic components - the type of each component is one of the following strings:
  - SAHRA
  - BINGIL
  - AMZGA

Input

Each request is on separate line. Accept requests until you get "STOPAAJUHIT!". The requests are the following:

- **"NEW *Name*"** - creates a new PLOKIJUH with the given name and price 0.0. If that one exists print "STUPIDO!!1"

  In the following requests the PLOKIJUHs will always exist:

- **"GEPRISEN *Name Price*"** - sets the price of the given PLOKIJUH
- **"KOMPONENTUNG *Name Component*"** - add a bozatronic component to the given PLOKIJUH. If the given component is different than the given print **"BLAH!"**
- **"DECOMPING *Name Component*"** - removes the first bozatronic component with the given name from the given PLOKIJUH. If the given component is different than the defined or cannot be found in the current PLOKIJUH print **"YACK!"**
- **"BLUSS *Name1 Name2*"** - creates a new PLOKIJUH with name that starts with the first half of the name of the first PLOKIJ and ends with the second half of the name of the second PLOKIJUH, price - the average of the prices of the given PLOKIJUHs and bozatronic components - all the components from the given PLOKIJUHs and destroys the given ones
- **"TULOSTASE"** - print all the PLOKIJUHs ordered ascending by the price as first criteria and descending by the count of bozatronic component as second criteria. The components for each one of them are ordered in the same order they've been added/ If there are no PLOKIJUHs print **"INTEPLOKIJUHAR:("**

Output

For every TULOSTATSE command print all the existing PLOKIJUHs in the following format:

$$# PLOKIJUH: Name
$#$ Price: *Price rounded 1 digit after the decimal point*
#$$ Components: *Component1*, *Component2*, ...
...

Constrains

- Allowed working time: 1 sec.
- Allowed memory consumption: 16 MB

Examples

Input	Output
TULOSTASE	INTEPLOKIJUHAR:(
NEW nqikoisi	STUPIDO!!1
GEPRISEN nqikoisi 3.14	BLAH!
KOMPONENTUNG nqikoisi AMZGA	YACK!
KOMPONENTUNG nqikoisi AMZGA	$$# PLOKIJUH: nqkoqsi
KOMPONENTUNG nqikoisi BINGIL	$#$ Price: 0.0
DECOMPING nqikoisi AMZGA	#$$ Components: AMZGA, SAHRA
NEW nqikoisi	$$# PLOKIJUH: nqikoisi
KOMPONENTUNG nqikoisi NQMAME	$#$ Price: 3.1
DECOMPING nqikoisi SAHRA	#$$ Components: AMZGA, BINGIL
NEW nqkoqsi	$$# PLOKIJUH: nqikoqsi
KOMPONENTUNG nqkoqsi AMZGA	$#$ Price: 1.6
KOMPONENTUNG nqkoqsi SAHRA	#$$ Components: AMZGA, BINGIL, AMZGA, SAHRA
TULOSTASE	
BLUSS nqikoisi nqkoqsi	
TULOSTASE	
STOPAAJUHIT!	