## Problem 1 – Activation Keys

Submit your solutions in the SoftUni Judge system [here](https://alpha.judge.softuni.org/contests/05-programming-fundamentals-final-exam/2302/practice" \l "1)

*You are about to make some good money, but first, you need to think of a way to verify who paid for your product and who didn't. You have decided to let people use the software for a free trial period and then require an activation key to continue using the product. Before you can cash out, the last step is to design a program that creates unique activation keys for each user. So, waste no more time and start typing!*

The first line of the input will be your raw activation key. It will consist of **letters and numbers only**.

After that, until the "Generate" command is given, you will be receiving strings with instructions for different operations that need to be performed upon the raw activation key.

There are several types of instructions, split by ">>>":

* "Contains>>>{substring}":
  + If the raw activation key contains the given substring, print  
     "{raw activation key} contains {substring}".
  + Otherwise, print "Substring not found!"
* "Flip>>>Upper/Lower>>>{startIndex}>>>{endIndex}":
  + Changes the substring **between the given indices (the end index is exclusive)** to upper or lower case and then prints the activation key.
  + All given indexes will be valid.
* **"Slice>>>{startIndex}>>>{endIndex}**":
  + **Deletes** the characters between the start and end indices (**the end index is exclusive) and** prints the activation key.
  + Both indices will be **valid**.

### Input

* The first line of the input will be a string consisting of **letters and numbers only**.
* After the first line, until the "Generate" command is given, you will be receiving **strings**.

### Output

* After the "Generate" command is received, print:
  + "Your activation key is: {activation key}"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| abcdefghijklmnopqrstuvwxyz  Slice>>>2>>>6  Flip>>>Upper>>>3>>>14  Flip>>>Lower>>>5>>>7  Contains>>>def  Contains>>>deF  Generate | abghijklmnopqrstuvwxyz  abgHIJKLMNOPQRstuvwxyz  abgHIjkLMNOPQRstuvwxyz  Substring not found!  Substring not found!  Your activation key is: abgHIjkLMNOPQRstuvwxyz |
| **Comments** | |
| 1. **Slice>>2>>6**   abcdefghijklmnopqrstuvwxyz becomes abghijklmnopqrstuvwxyz   1. **Flip>>>Upper>>>3>>>14**   abghijklmnopqrstuvwxyz becomes abgHIJKLMNOPQRstuvwxyz   1. **Flip>>>Lower>>>5>>>7**   abgHIJKLMNOPQRstuvwxyz becomes abgHIjkLMNOPQRstuvwxyz   1. **Contains>>>def**   abgHIjkLMNOPQRstuvwxyz does not contain def   1. **Contains>>>deF**   abgHIjkLMNOPQRstuvwxyz does not contain deF  The final activation key is abgHIjkLMNOPQRstuvwxyz | |
| **Input** | **Output** |
| 134softsf5ftuni2020rockz42  Slice>>>3>>>7  Contains>>>-rock  Contains>>>-uni-  Contains>>>-rocks  Flip>>>Upper>>>2>>>8  Flip>>>Lower>>>5>>>11  Generate | 134sf5ftuni2020rockz42  Substring not found!  Substring not found!  Substring not found!  134SF5FTuni2020rockz42  134SF5ftuni2020rockz42  Your activation key is: 134SF5ftuni2020rockz42 |

## Problem 2 – Emoji Detector

Submit your solutions in the SoftUni Judge system [here](https://alpha.judge.softuni.org/contests/05-programming-fundamentals-final-exam/2302/practice" \l "2)

[Your task is to write a program that extracts em](https://alpha.judge.softuni.org/contests/05-programming-fundamentals-final-exam/2302/practice" \l "2)ojis from a text and finds the threshold based on the input.

You have to get your C**ool threshold**. It is obtained by **multiplying all** the digits found in the input. The cool threshold could be a **huge number**, so be mindful.

An emoji is valid when:

* It is surrounded by 2 characters, either "::" or "\*\*".
* It is **at least 3** characters long (**without** the surrounding symbols).
* **It starts** with a **capital letter.**
* Continues with **lowercase** letters **only.**

Examples of valid emojis: ::Joy::, \*\*Banana\*\*, ::Wink::

Examples of invalid emojis: ::Joy\*\*, **::fox:es:**, **\*\*Monk3ys\*\*, :Snak::Es::**

You need to count **all valid emojis** in the text and calculate their **coolness**. The coolness of the emoji is **determined** by summing all the **ASCII values of all letters** in the emoji.

Examples: ::Joy:: - 306, \*\*Banana\*\* - 577, ::Wink:: - 409

You need to print the result of the cool threshold and after that – to take all emojis out of the text, count them and print **only the cool ones** on the console.

### Input

* On the single input, you will receive a piece of string.

### Output

* On the first line of the output, print the obtained Cool threshold in the format:

**"Cool threshold: {coolThresholdSum}"**

* On the following line, **print the** **count of all emojis** found in the text in format:

"{countOfAllEmojis} emojis found in the text. The cool ones are:

{cool emoji 1}

{cool emoji 2}

…

{cool emoji N}"

### Constraints

There will always be at least one digit in the text!

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| In the Sofia Zoo there are 311 animals in total! ::Smiley:: This includes 3 \*\*Tigers\*\*, 1 ::Elephant:, 12 \*\*Monk3ys\*\*, a \*\*Gorilla::, 5 ::fox:es: and 21 different types of :Snak::Es::. ::Mooning:: \*\*Shy\*\* | Cool threshold: 540  4 emojis found in the text. The cool ones are:  ::Smiley::  \*\*Tigers\*\*  ::Mooning:: |
| **Comments** | |
| You can see all the valid emojis in green. There are various reasons why the rest are not valid, examine them carefully. The "cool threshold" is 3\*1\*1\*3\*1\*1\*2\*3\*5\*2\*1 = 540.  ::Smiley:: -> 83 + 109 + 105 + 108 + 101 + 121 = 627 > 540 -> cool  \*\*Tigers\*\* -> 84 + 105 + 103 + 101 + 114 + 115 = 622 > 540 -> cool  ::Mooning:: -> 77 + 111 + 111 + 110 + 105 + 110 + 103 = 727 > 540 -> cool  \*\*Shy\*\* -> 83 + 104 + 121 = 308 < 540 -> not cool  In the end, we print the count of all valid emojis found and each of the cool ones on a new line. | |
| **Input** | **Output** |
| 5, 4, 3, 2, 1, go! The 1-th consecutive banana-eating contest has begun! ::Joy:: \*\*Banana\*\* ::Wink:: \*\*Vali\*\* ::valid\_emoji:: | Cool threshold: 120  4 emojis found in the text. The cool ones are:  ::Joy::  \*\*Banana\*\*  ::Wink::  \*\*Vali\*\* |
| **Input** | **Output** |
| It is a long established fact that 1 a reader will be distracted by 9 the readable content of a page when looking at its layout. The point of using ::LoremIpsum:: is that it has a more-or-less normal 3 distribution of 8 letters, as opposed to using 'Content here, content 99 here', making it look like readable \*\*English\*\*. | Cool threshold: 17496  1 emojis found in the text. The cool ones are: |
| **Comments** | |
| You can see \*\*English\*\* is a valid emoji, but the sum of ASCII **is not** **bigger** than the cool threshold. That's why we **don't** print anything in the end. | |

## Problem 3 - Heroes of Code and Logic VII

Submit your solutions in the SoftUni Judge system [here](https://alpha.judge.softuni.org/contests/04-programming-fundamentals-final-exam/2303/practice#30)

*You got your hands on the most recent update on the best MMORPG of all time – Heroes of Code and Logic. You want to play it all day long! So cancel all other arrangements and create your party!*

On the first line of the standard input, you will receive an integer **n** – the number of heroes that you can choose for your party. On the next **n** lines, the heroes themselves will follow with their **hit points** and **mana points** separated by a single space in the following format:

"{hero name} {HP} {MP}"

* HP stands for hit points and MP for mana points
* a hero can have a maximum of 100 HP and 200 MP

After you have successfully picked your heroes, you can start playing the game. You will be receiving different commands, each on a new line, separated by " – ", until the "End" command is given.

There are several actions that the heroes can perform:

"CastSpell – {hero name} – {MP needed} – {spell name}"

* If the hero has the required MP, he casts the spell, thus reducing his MP. Print this message:
  + "{hero name} has successfully cast {spell name} and now has {mana points left} MP!"
* If the hero is unable to cast the spell print:
  + "**{hero name} does not have enough MP to cast {spell name}!**"

"TakeDamage – {hero name} – {damage} – {attacker}"

* Reduce the hero HP by the given damage amount. If the hero is still alive (his HP is greater than 0) print:
  + "{hero name} was hit for {damage} HP by {attacker} and now has {current HP} HP left!"
* If the hero has died, remove him from your party and print:
  + "{hero name} has been killed by {attacker}!"

"Recharge – {hero name} – {amount}"

* The hero increases his MP. If it brings the MP of the hero above the **maximum value** (**200)**, MP is increased to **200**. (the MP can't go over the maximum value).
* Print the following message:
  + "{hero name} recharged for {amount recovered} MP!"

"Heal – {hero name} – {amount}"

* The hero increases his HP. If a command is given that would bring the HP of the hero above the **maximum value (100)**, HP is increased to **100** (the HP can't go over the maximum value).
* Print the following message:
  + "{hero name} healed for {amount recovered} HP!"

### Input

* On the first line of the standard input, you will receive an integer **n.**
* On the following **n** lines, the heroes themselves will follow with their **hit points** and **mana points** separated by a space in the following format
* You will be receiving different **commands**, each on a new line, separated by " – ", until the "End" command is given.

### Output

* Print all members of your party who are **still alive** in the following format (their HP/MP need to be indented 2 spaces):

"{hero name}

HP: {current HP}

MP: {current MP}"

### Constraints

* The starting HP/MP of the heroes will be valid, 32-bit integers will never be negative or exceed the respective limits.
* The HP/MP amounts in the commands will never be negative.
* The hero names in the commands will always be valid members of your party. No need to check that explicitly.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  Solmyr 85 120  Kyrre 99 50  Heal - Solmyr - 10  Recharge - Solmyr - 50  TakeDamage - Kyrre - 66 - Orc  CastSpell - Kyrre - 15 - ViewEarth  End | Solmyr healed for 10 HP!  Solmyr recharged for 50 MP!  Kyrre was hit for 66 HP by Orc and now has 33 HP left!  Kyrre has successfully cast ViewEarth and now has 35 MP!  Solmyr  HP: 95  MP: 170  Kyrre  HP: 33  MP: 35 |
| **Input** | **Output** |
| 4  Adela 90 150  SirMullich 70 40  Ivor 1 111  Tyris 94 61  Heal - SirMullich - 50  Recharge - Adela - 100  CastSpell - Tyris - 1000 - Fireball  TakeDamage - Tyris - 99 - Fireball  TakeDamage - Ivor - 3 - Mosquito  End | SirMullich healed for 30 HP!  Adela recharged for 50 MP!  Tyris does not have enough MP to cast Fireball!  Tyris has been killed by Fireball!  Ivor has been killed by Mosquito!  Adela  HP: 90  MP: 200  SirMullich  HP: 100  MP: 40 |
| **Comments** | | |
| Heal – SirMullich healed for 30 HP due to the HP max limit.  Recharge – Adela recharged for 50 MP due to the MP max limit.  CastSpell – Tyris does not have enough MP to cast the spell.  TakeDamage – Tyris's HP is reduced by 99, thus becoming -5, which means he is dead.  TakeDamage – Ivor's HP is now -2, so he is dead too.  After the "End" command, we print the remaining living heroes, sorted by their HP in reverse order. | | |