# Exercises: Strings and Text Processing

Problems for exercises and homework for the [“Programming Fundamentals Extended” course @ SoftUni](https://softuni.bg/courses/programming-fundamentals).

Check your solutions here: <https://judge.softuni.bg/Contests/441>.

## Find the Letter

Write a program, which receives a string and prints the index of a given letter in the string. The tricky part is that you will have to find **not** the first letter, but the **nthletter**.

### Input

* On the first line, you will receive the **string** you are going to search through.
* On the **second** line, you will receive an **array** with exactly **two** **elements**:
  + The **first** element will be the **letter**, which you have to search for.
  + The **second** element will be an **integer N**, showing us which **occurrence** ofthe **letter** we are searching for.

**Example**: If we receive the string “Programming is awesome!” and on the next line we receive the array “m 3”. We should find the **third** occurrence of the letter ‘**m**’. It can be found on **20th** index.

### Output

If the nth occurrence of the letter is present in the string, print **the index** of that occurrence.

If the letter is **not present** in the string, or there are **less occurrences** than **N**, print:

* “Find the letter yourself!”

*Note: the* ***comparison*** *should be* ***case******sensitive****.*

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Programming is awesome!  m 3 | 20 |
| Strings, strings everywhere...  e 5 | Find the letter yourself! |

## Placeholders

Placeholders in C# are really comfortable for use, right? How about you implement them to see just how easy they are.

### Input

You will be given input lines containing a string with placeholders, i.e. “This is {0}. And that is {1}.”.

The input lines will be in the following format:

{string} -> {element}, {element2}

You should **replace** the **placeholders** in the **string** with the **elements** given **after** the“ -> ” **delimiter**. The elements are **separated** by “, ”. **Each** **element** has an **index**, so you must give them the **right order**. The **first given element** goes where the **0** is, the **second** to the **1**, and so on…

### Output

Print **each** **string** with its **replaced** **placeholders**, right **after** you’ve read it, and **before** reading the next one.

The input ends when you receive the command “**end**”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| I have a {0} and a {1} -> car, house  Cool, and I have a {0}. -> yacht  Darn... You beat me {0} {1} -> with, this  end | I have a car and a house  Cool, and I have a yacht.  Darn... You beat me with this |
| This is {2} {1} {0}. -> ment, I, what  And this is what you ment. -> nothing  end | This is what I ment.  And this is what you ment. |

### Hints

* Since the sentences hold spaces, you cannot split the sentence and the values by space.  
  There should be some functionality in C#, which removes the border spaces from a string, if you split it by   
  “->”, though.
* You might have to **split** the **sentence** and the **values first**, and **then the values**, because their **delimiter** **contains** a **space**.

## JSON Stringify

JSON is a syntax for storing and exchanging data. It’s derived from **J**ava**S**cript **O**bject **N**otation. You have been tasked to create a database of students, and store them in the JSON format.

First, create a **Student class**, which will represent our **data** **model** – it will hold our **data**. The **Student** should have a **Name** (**string**), an **Age** (**integer**), and **Grades** (**Collection** of **integer numbers**).

This will be enough to store our data. We’ll need a collection of **Students**, since we will store a lot of students.

And now let’s see the input and output formats.

### Input

The input will consist of several input lines in the following format:

{name} : {age} -> {grade1}, {grade2}, {grade3}. . .

Store every student with their **name**, **age** and **given grades**. The **name** will be a **string** which can contain **any** **ASCII** characters **except** **space** (“ “), “:”, “-”, “>” and “,”. The **age** will be an **integer**. The **grades** will be **integers**.

When you receive the command “**stringify**”, the input sequence ends.

### Output

You should print the **collection of students** in the following format: [{student1},{student2}. . .]

Each student must be printed in the following format:

* {name:"{name}",age:{age},grades:[{grade1}, {grade2}. . .]}

See the examples for more info.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho : 25 -> 6, 6, 5  Gosho : 10 -> 3, 3, 4  Ivan : 28 -> 2, 2, 3  stringify | [{name:"Pesho",age:25,grades:[6, 6, 5]},{name:"Gosho",age:10,grades:[3, 3, 4]},{name:"Ivan",age:28,grades:[2, 2, 3]}] |
| Johnny : 11 -> 6, 4, 3  Peter : 10 -> 5, 5, 4  Jordan : 13 -> 6, 6  Donald : 10 ->  Isacc : 20 -> 2, 2, 3  Alex : 11 -> 6, 6, 3  stringify | [{name:"Johnny",age:11,grades:[6, 4, 3]},{name:"Peter",age:10,grades:[5, 5, 4]},{name:"Jordan",age:13,grades:[6, 6]},{name:"Donald",age:10,grades:[]},{name:"Isacc",age:20,grades:[2, 2, 3]},{name:"Alex",age:11,grades:[6, 6, 3]}] |

## JSON Parse

Stringifying a JSON was easy? You want something harder?

Well… JSON parse is your problem. You will receive a JSON collection of students. You need to parse it to Student objects.

### Input

The input will consist of a single line containing the students’ info. The students will be given in the following format:

* [{student1},{student2}. . .]

Each student will be given in the following format:

* {name:"{name}",age:{age},grades:[{grade1}, {grade2}. . .]}

### Output

Parse the input data and print each student in the following format:

* {name} : {age} -> {grade1}, {grade2}, {grade3}...

In cases where a student has **NO grades**, print “None”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [{name:"Pesho",age:25,grades:[6, 6, 5]},{name:"Gosho",age:10,grades:[3, 3, 4]},{name:"Ivan",age:28,grades:[2, 2, 3]}] | Pesho : 25 -> 6, 6, 5  Gosho : 10 -> 3, 3, 4  Ivan : 28 -> 2, 2, 3 |
| [{name:"Johnny",age:11,grades:[6, 4, 3]},{name:"Peter",age:10,grades:[5, 5, 4]},{name:"Jordan",age:13,grades:[6, 6]},{name:"Donald",age:10,grades:[]},{name:"Isacc",age:20,grades:[2, 2, 3]},{name:"Alex",age:11,grades:[6, 6, 3]}] | Johnny : 11 -> 6, 4, 3  Peter : 10 -> 5, 5, 4  Jordan : 13 -> 6, 6  Donald : 10 -> None  Isacc : 20 -> 2, 2, 3  Alex : 11 -> 6, 6, 3 |

## Points Counter

Write a program, which receives data about a **team**, **player** and **points**.

### Input

You can have **two** types of input:

* {TEAM}|{Player}|{points}
* {Player}|{TEAM}|{points}

The **team** **name** will always consist of **uppercase** **Latin** **letters** and the player name will always **start** with **uppercase** **Latin** **letter** and **all** **other** **letters** will be **lowercase**.

The **team** and **player** names **might** be **polluted** with some **prohibited** symbols (‘**@**’, ‘**%**’, ‘**$**’ and ‘**\***’). You have to **delete** **every** **occurrence** of these symbols in **every** **team** and **player** name.

Then, calculate every team’s total score. Every **team’s** **total** **score** equals to the **total** **sum** of the **points** made by **every** **player** **in** the **team**.

### Output

When you receive the command “**Result**”, print **all** teams, ordered in **descending order** by the **points** they made and the player with **most** points in the **team** in the following format:

|  |
| --- |
| {teamName} => {totalSumOfPoints}  Most points scored by: {nameOfThePlayer} |

In case of **repeating** **player** **names** for one team, **save** the **value**, which is **received** **last**.

### Constraints

* The **team** **names** will be **at least** **2** characters **long**
* The **points** for each **player** will be in the interval [1…100]

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| LA|Bryant|70  L%@A|Odom|67  James|%CAVA@@LIE$$$RS|54  C@art%er|GR%%IZZ%%LIE@S@@@|49  Anthony|KNICKS|11  UTAH|Jo%%%%hn$$so@@n|24  S@@PU\*R\*S$|Ga\*\*\*so\*\*l|32  Jone@@@@s|KNICKS|5  Result | LA => 137  Most points scored by Bryant  CAVALIERS => 54  Most points scored by James  GRIZZLIES => 49  Most points scored by Carter  SPURS => 32  Most points scored by Gasol  UTAH => 24  Most points scored by Johnson  KNICKS => 16  Most points scored by Anthony |
| SO@@@FTU%\*NI|P\*&@esho|30  SO$$FT\*UNI|Gos%@ho|42  PAARTHURNAX|Maria|35  S\*OFT$$$UNI|Iv\*\*\*\*@an|3  L@u\*b%o@|HE\*\*\*RO@@ES|11  Result | SOFTUNI => 75  Most points scored by Gosho  PAARTHURNAX => 35  Most points scored by Maria  HEROES => 11  Most points scored by Lubo |

## Sentence Split

Splitting the input by a given delimiter is easy, especially when you are told in the problem descriptions, that the elements will **not** contain the delimiter. What if, however, they do.

### Input

You will be **given a sentence**, which may contain **any ASCII character**.  
Then you will be **given** a **delimiter**, which may **also** contain **any ASCII character**.

### Output

Your task is to **split the sentence** by the **given** **delimiter**, and print each of the elements, in the following format:

* [{element1}, {element2}, {element3}. . .]

See the examples for more info.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| This will be hard, right. Nope!  . ***(dot and space)*** | [This will be hard, right, Nope!] |
| Bau, Chika, Bow Wow  , ***(comma and space)*** | [Bau, Chika, Bow Wow] |

## Capitalize Words

Create a program, which receives sentences of words, and capitalizes the words in them.

The sentences will be **grammatically correct**. You must **extract** **each** **word** from the sentences, and you must **capitalize** it.

The **capitalization** of a **word** is the process of making the **first letter** **Capital**, and **every other letter** – **lowercase**.

You should read input lines, until you receive the command “**end**”.

You **might** receive **SEVERAL sentences** on **1 input line**.

After you’ve read a sentence, you extract the words, **capitalize them**, and **print them**, **separated** by a **space** and a **comma**. **ONLY** then should you **read** the **next sentence**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| This is funny.  This is not.  end | This, Is, Funny  This, Is, Not |
| Let's... gEt... iT RoCKing In HeRE!!!  PartY rock ANTHEM! ! !  end | Let's, Get, It, Rocking, In, Here  Party, Rock, Anthem |

### Hints

* The sentences can contain all kinds of **non-letter** characters. Try splitting by all kinds of **punctuation**.