# More Exercises: Strings and Text Processing

Problems for exercises and homework for the ["Technology Fundamentals" course @ SoftUni](https://softuni.bg/courses/technology-fundamentals).

You can check your solutions here: <https://judge.softuni.bg/Contests/1332>

## Extract Person Information

Write a program that reads **n** lines of strings and extracts the **name** and **age** of a given person. The name of the person will be **between '@'** and **'|'**. The person’s **age** will be **between '#'** and **'\*'**. **Example: "Hello my name is @Peter| and I am #20\* years old." For each** found name and age **print** a line in the following format **"{name} is {age} years old."**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  Here is a name @George| and an age #18\*  Another name @Billy| #35\* is his age | George is 18 years old.  Billy is 35 years old. |
| 3  random name @lilly| random digits #5\* age  @Marry| with age #19\*  here Comes @Garry| he is #48\* years old | lilly is 5 years old.  Marry is 19 years old.  Garry is 48 years old. |

## Ascii Sumator

Write a program that prints a **sum of all characters between two given characters** (their **ascii code**). On the **first line** you will get a **character**. On the **second line** you get **another character**. On the **last line** you get a **random string**. Find all the characters **between the two given** and **print their ascii sum**.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| .  @  dsg12gr5653feee5 | 363 |
| ?  E  @ABCEF | 262 |

## \*\*Treasure Finder

Write a program that **decrypts a message** by a given **key** and gathers information about hidden **treasure type** and its **coordinates.** On the **first line** you will receive a **key (sequence of numbers).** On the **next few lines until you receive "find"** you will get lines of **strings**. You have to **loop through every string** and **decrease the ascii code of each character** with a **corresponding number of the key** sequence. The way you choose a key number from the sequence is just **looping through it**. If the **length of the key** sequence is **less than the string** sequence, you start **looping from the beginning of the key.** For more clarification see the example below. **After decrypting** the message you will **get a type of treasure and its coordinates.** The **type** will be **between** the symbol **'&'** and the coordinates will be between the symbols **'<'** and **'>'**. For each line **print the type and the coordinates** in format **"Found {type} at {coordinates}".**

### Example

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 1 2 1 3  ikegfp'jpne)bv=41P83X@  ujfufKt)Tkmyft'duEprsfjqbvfv=53V55XA  find | Found gold at 10N70W  Found Silver at 32S43W | We start looping through the first string and the key. When we reach the end of the key we start looping from the beginning of the key, but we continue looping through the string. (until the string is over)  The first message is: **"hidden&gold&at<10N70W>"** so we print we found gold at the given coordinates  We do the same for the second string  **"thereIs&Silver&atCoordinates<32S43W>"**(starting from the beginning of the key and the beginning of the string) |

## \*Morse Code Translator

Write a program that translates messages from **Morse code to English** (**capital letters).** Use [this](https://morsecode.scphillips.com/morse2.html) page to help you (**without the numbers**). The words will be separated by a **space (' ')**. There will be a **'|'** character which you should **replace with ' '** (space).

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| .. | -- .- -.. . | -.-- --- ..- | .-- .-. .. - . | .- | .-.. --- -. --. | -.-. --- -.. . | I MADE YOU WRITE A LONG CODE |
| .. | .... --- .--. . | -.-- --- ..- | .- .-. . | -. --- - | -- .- -.. | I HOPE YOU ARE NOT MAD |

## HTML

You will receive **3 lines** of input. On the **first line** you will receive a **title of an article**. On the **next line** you will receive the **content of that article**. On the next **n** lines until you receive **"end of comments"** you will get the **comments about the article**. Print the **whole information in html format**. The **title** should be in **"h1" tag (<h1></h1>);** the **content** in **article tag (<article></article>);** each **comment** should be in **div tag (<div></div>).** For more clarification see the example below

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| SoftUni Article  Some content of the SoftUni article  some comment  more comment  last comment  end of comments | <h1>  SoftUni Article  </h1>  <article>  Some content of the SoftUni article  </article>  <div>  some comment  </div>  <div>  more comment  </div>  <div>  last comment  </div> |

## \*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 '@'. |

'

## \*Winning Ticket

Lottery is exciting. What is not, is checking a million tickets for winnings only by hand. So, you are given the task to create a program which automatically checks if a ticket is a winner.

You are given a **collection of tickets separated by commas and spaces**. You need to check every one of them if it has a winning combination of symbols.

**A valid ticket should have exactly 20 characters**. The winning symbols are '**@**', '**#**', '**$**' and '**^**'. But in order for a ticket to be a winner the symbol should uninterruptedly repeat for at least **6 times** in both the **tickets left half** and the **tickets right half**.

For example, a valid winning ticket should be something like this:

"**Cash$$$$$$Ca$$$$$$sh**"

The left half "**Cash$$$$$$**" contains "**$$$$$$**", which is also contained in the tickets right half "**Ca$$$$$$sh**". A winning ticket should contain symbols repeating up to 10 times in both halves, which is considered a Jackpot (for example: "**$$$$$$$$$$$$$$$$$$$$**").

**Input**

The input will be read from the console. The input consists of a **single line** containing all tickets **separated by commas and one or more white spaces** in the format:

* **"{ticket}, {ticket}, … {ticket}"**

**Output**

Print the result for every ticket in the order of their appearance, each on a separate line in the format:

* **Invalid ticket - "invalid ticket"**
* **No match - "ticket "{ticket}" - no match"**
* **Match with length 6 to 9 - "ticket "{ticket}" - {match length}{match symbol}"**
* **Match with length 10 - "ticket "{ticket}" - {match length}{match symbol} Jackpot!"**

**Constrains**

* Number of tickets will be in range [0 … 100]

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Cash$$$$$$Ca$$$$$$sh | ticket "Cash$$$$$$Ca$$$$$$sh" - 6$ |
| $$$$$$$$$$$$$$$$$$$$, aabb , th@@@@@@eemo@@@@@@ey | ticket "$$$$$$$$$$$$$$$$$$$$" - 10$ Jackpot!  invalid ticket  ticket "th@@@@@@eemo@@@@@@ey" - 6@ |
| validticketnomatch:( | ticket "validticketnomatch:(" - no match |