# More Exercises: Text Processing and Regular Expressions

Problems for exercises and homework for the [“Technology Fundamentals with C#” course @ SoftUni](https://softuni.bg/modules/57/tech-module-4-0).

You can check your solutions in [Judge](https://judge.softuni.bg/Contests/1338).

## 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 the '@' symbol** and the **'|' symbol**. 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 а hidden **treasure type** and its **coordinates.** On the **first line,** you will receive a **key (а 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 ' '** (а 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> |

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

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}"

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!"

**Example**

|  |  |
| --- | --- |
| **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 |

## Nether Realms

Mighty battle is coming. In the stormy nether realms, demons are fighting against each other for supremacy in a duel, from which only one will survive.

Your job, however is not so exciting. You are assigned to **sign in all the participants** in the nether realm's mighty battle's demon book, which of course is **sorted alphabetically**.

A demon's **name contains his health and his damage**.

The **sum of the asci codes** of **all characters** (excluding numbers (0-9), arithmetic symbols (**'+', '-', '\*', '/'**) and delimiter dot (**'.'**)) gives a **demon's total health**.

**The sum of all numbers** in his name forms his base damage. Note that you should consider the plus **'+'** and minus **'-'** signs (e.g. **+10 is 10** and **-10 is -10**). However, there are some symbols (**'\*'** and **'/'**) that can further **alter the base damage by multiplying or dividing it by 2** (e.g. in the name "m**15**\*/c**-5.0**", the base damage is **15 + (-5.0) = 10** and then you need to multiply it by 2 (e.g. 10 \* 2 = 20) and then divide it by 2 (e.g. 20 / 2 = 10)).

So, **multiplication and division** are applied **only after all numbers are included** in the calculation and **in the order they appear in the name**.

You will get all demons **on a single line**, separated by commas and zero or more blank spaces. Sort them in **alphabetical order** and print their names **along their health and damage**.

The input will be read from the console. The input consists of a **single line** containing all demon names **separated by commas and zero or more spaces** in the format: **"{demon name}, {demon name}, … {demon name}"**

Print all demons **sorted by their name in ascending order**, each on a separate line in the format:

* **"{demon name} - {health points} health, {damage points} damage"**

### Constraints

* A demon's name will contain **at least one character**
* A demon's name **cannot contain** blank spaces ' ' or commas ','
* A **floating point number will always have digits before and after its decimal separator**
* **Number** in a demon's name **is considere**d everything that is a valid integer or floating point number (with dot '.' used as separator). For example, all these are valid numbers: '4', '+4', '-4', '3.5', '+3.5', '-3.5'

### Example

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Comments** | |
| M3ph-0.5s-0.5t0.0\*\* | M3ph-0.5s-0.5t0.0\*\* - 524 health, 8.00 damage | M3ph-0.5s-0.5t0.0\*\*:  Health = 'M' + 'p' + 'h' + 's' + 't' = 524 health.  Damage = (3 + (-0.5) + (-0.5) + 0.0) \* 2 \* 2 = 8 damage. | |
| **Input** | **Output** | | **Comments** |
| M3ph1st0\*\*, Azazel | Azazel - 615 health, 0.00 damage  M3ph1st0\*\* - 524 health, 16.00 damage | | Azazel:  Health - 'A' + 'z' + 'a' + 'z' + 'e' + 'l' = 615 health. Damage - no digits = 0 damage.  M3ph1st0\*\*:  Health - 'M' + 'p' + 'h' + 's' + 't' = 524 health.  Damage - (3 + 1 + 0) \* 2 \* 2 = 16 damage. |
| Gos/ho | Gos/ho - 512 health, 0.00 damage | |  |