Exercises: Regular Expressions

This document defines the exercises for "Java Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

Problem 1. Match Full Name

Write a regular expression to match a valid full name. A valid full name consists of **two words**, each word **starts** with a **capital letter** and contains **only lowercase letters afterwards**; each word should be **at least two letters long**; the two words should be **separated by a single space**.

To help you out, we've outlined several steps:

- Use an online regex tester like https://regex101.com/
- Check out how to use **character sets** (denoted with square brackets - "[]")
- Specify that you want two words with a space between them (the **space character ' '**, and not any whitespace symbol)
- For each word, specify that it should begin with an uppercase letter using a **character set**. The desired characters are in a range – **from 'A' to 'Z'**.
- For each word, specify that what follows the first letter are only **lowercase letters**, one or more – use another character set and the correct **quantifier**.
- To prevent capturing of letters across new lines, put "\b" at the beginning and at the end of your regex. This will ensure that what precedes and what follows the match is a word boundary (like a new line).

In order to check your regex, use these values for reference (paste all of them in the Test String field):

Examples

Match ALL of these	Match NONE of these
`Ivan Ivanov`	`ivan ivanov, Ivan ivanov, ivan Ivanov, IVan Ivanov, Ivan IvAnov, Ivan Ivanov`

Input	Output

`ivan ivanov`
`Ivan Ivanov`
`end`
	`Ivan Ivanov`

Problem 2. Match Phone Number

Write a regular expression to match a valid phone number from Sofia. A valid number will start with "+359" followed by the area code (2) and then the number itself, consisting of 7 digits (separated in two group of 3 and 4 digits respectively). The different parts of the number are separated by **either a space or a hyphen** ('-'). Refer to the examples to get the idea.

- Use quantifiers to match a specific number of digits
- Use a capturing group to make sure the delimiter is **only one of the allowed characters (space or hyphen)** and not a combination of both. Use the group number to achieve this

- Add a word boundary at the end of the match to avoid partial matches (the last example on the right-hand side)
- Ensure that before the '+' sign there is either a space or the beginning of the string

Examples

Match ALL of these	Match NONE of these
+359 2 222 2222 +359-2-222-2222	359-2-222-2222, +359/2/222/2222, +359-2 222 2222 +359 2-222-2222, +359-2-222-222, +359-2-222-22222

Input	Output
+359 2 222 2222 +3591345123 end	+359 2 222 2222
+359 2 234 5678 +359-2-234-5678 +359-2 234-5678 end	+359 2 234 5678 +359-2-234-5678

Problem 3. Series of Letters

Write a program that reads a string from the console and replaces all series of consecutive identical letters with a single one.

Examples

Input	Output
aaaaabbbbbcdddeeeedssaa	abcdedsa

Problem 4. Replace <a> tag

Write a program that replaces in a HTML document given as string **all the tags …** with corresponding **tags [URL href=…>…[/URL]**. Read an input, until you receive **"end" command**. Print the result on the console.

Examples

Input	Output
SoftUni **end**	 [URL href=**http://softuni.bg**>SoftUni[/URL]

Note: The input may be read on a single line (unlike the example above) or from a file. Remove all new lines if you choose the first approach.

Problem 5. Extract Emails

Write a program to **extract all email addresses from a given text**. The text comes at the only input line. Print the emails on the console, each at a separate line. Emails are considered to be in format **<user>@<host>**, where:

- **<user>** is a sequence of letters and digits, where '**.**', '**-**' and '**_**' can appear between them. Examples of valid users: "**stephan**", "**mike03**", "**s.johnson**", "**st_steward**", "**softuni-bulgaria**", "**12345**". Examples of invalid users: "**--123**", ".....", "**nakov_-**", "**_steve**", "**.info**".
- **<host>** is a sequence of at least two words, separated by dots '**.**'. Each word is sequence of letters and can have hyphens '**-**' between the letters. Examples of hosts: "**softuni.bg**", "**software-university.com**", "**intoprogramming.info**", "**mail.softuni.org**". Examples of invalid hosts: "**helloworld**", "**.unknown.soft.**", "**invalid-host-**", "**invalid-**".
- Examples of **valid emails**: info@softuni-bulgaria.org, kiki@hotmail.co.uk, no-reply@github.com, s.peterson@mail.uu.net, info-bg@software-university.software.academy.
- Examples of **invalid emails**: --123@gmail.com, …@mail.bg, .info@info.info, _steve@yahoo.cn, mike@helloworld, mike@.unknown.soft., s.johnson@invalid-.

Examples

Input	Output
`Please contact us at: support@github.com.`	*support@github.com*
`Just send email to s.miller@mit.edu and j.hopking@york.ac.uk for more information.`	*s.miller@mit.edu* *j.hopking@york.ac.uk*
`Many users @ SoftUni confuse email addresses. We @ Softuni.BG provide high-quality training @ home or @ class. -- steve.parker@softuni.de.`	*steve.parker@softuni.de*

Problem 6. Sentence Extractor

Write a program that reads a **keyword** and some **text** from the console and prints **all sentences from the text, containing that word**. A sentence is any sequence of words ending with '**.**', '**!**' or '**?**'.

Examples

Input	Output
`is` `This **is** my cat! And this **is** my dog. We happily live in Paris – the most beautiful city in the world! Isn't it great? Well it is :)`	`This is my cat!` ` And this is my dog.`

Problem 7. * Valid Usernames

This problem is from the Java Basics Exam (21 September 2014 Evening). You may check your solution here.

You are part of the back-end development team of the next Facebook. You are given **a line of usernames**, between one of the following symbols**: space, "/", "\", "(", ")".** First you have to export all **valid** usernames. A valid username **starts with a letter** and can contain **only letters, digits and "_"**. It cannot be **less than 3 or more than 25 symbols** long. Your task is to **sum** the length of **every 2 consecutive valid** usernames and print on the console the 2 valid usernames with **biggest sum** of their **lengths,** each on a separate line.

Input

The input comes from the console. One line will hold all the data. It will hold **usernames,** divided by the symbols: **space, "/", "\", "(", ")".**

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

Print at the console the 2 **consecutive valid usernames** with the **biggest sum** of their lengths each on a separate line. If there are **2 or more couples** of usernames with the same sum of their lengths, print he **left most**.

Constraints

- The input line will hold characters in the range [0 … 9999].
- The usernames should **start with a letter** and can contain **only letters, digits and "_"**.
- The username cannot be **less than 3 or more than 25 symbols** long.
- Time limit: 0.5 sec. Memory limit: 16 MB.

Examples

Input	Output
`ds3bhj y1ter/wfsdg 1nh_jgf ds2c_vbg\4htref`	`wfsdg` `ds2c_vbg`

Input	Output
`min23/ace hahah21 (  sasa  )  att3454/a/a2/abc`	`hahah21` `sasa`

Input	Output
`chico/ gosho \ sapunerka (3sas) mazut  lelQ_Van4e`	`mazut` `lelQ_Van4e`

Problem 8. ** Extract Hyperlinks

This problem is originally from the JavaScript Basics Exam (27 July 2014). You may check your solution here.

Write a program to **extract all hyperlinks** (**<href=…>**) from a given text. The text comes from the console on a variable number of lines and ends with the command "END". Print at the console the **href** values in the text.

The input text is **standard HTML code**. It may hold many tags and can be formatted in many different forms (with or without whitespace). The **<a>** elements may have many attributes, not only **href**. You should extract only the values of the **href** attributes of all **<a>** elements.

Input

The input data comes from the console. It ends when the "END" command is received.

Output

Print at the console the **href** values in the text, each at a separate line, in the order they come from the input.

Constraints

- The input will be **well formed HTML fragment** (all tags and attributes will be correctly closed).
- Attribute values will never hold tags and hyperlinks, e.g. "**<img alt='<a href="hello">' />**" is invalid.
- Commented links are also extracted.
- The number of input lines will be in the range [1 ... 100].
- Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
 END	http://softuni.bg
This text has no links END	
<!DOCTYPE html> <html> <head> <title>Hyperlinks</title> <link href="theme.css" rel="stylesheet" /> </head> <body> Home Courses Forum Forum click nak href='fake'<a href="#"> <!-- This code is commented: commentex hyperlink --> </body> END	/ /courses /forum # javascript:alert('hi yo') http://www.nakov.com #empty # #commented

Problem 9. * Query Mess

This problem is originally from the JavaScript Basics Exam (22 November 2014). You may check your solution here.

Ivancho participates in a team **project** with colleagues at **SoftUni**. They have to develop **an application**, but something *mysterious* happened – at the last moment all team members… disappeared! And guess what? He is left **alone** to finish the project. All that is left to do is to parse the input data and store it in a special way, but Ivancho has no idea how to do that! Can you help him?

Input

The input comes from the console on a variable number of lines and ends when the keyword "END" is received.

For each row of the input, the query string contains pairs field=value. Within each pair, the field name and value are separated by an equals sign, '='. The series of pairs are separated by an ampersand, '&'. The question mark is used as a separator and is not part of the query string. A URL query string may contain another URL as value. The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

For each input line, print on the console a line containing the **processed string as follows**:
key=[value]nextkey=[another value] … etc.

Multiple whitespace characters should be reduced to one inside value/key names, but there shouldn't be any whitespaces before/after extracted **keys** and **values**. If a key **already exists**, the value is added with comma and space after other values of the existing key in the current string. See the **examples** below.

Constraints

- SPACE is encoded as '+' or "%20". Letters (A-Z and a-z), numbers (0-9), the characters '*', '-', '.', '_' and *other non-special symbols* are left as-is.
- Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
login=student&password=student END	login=[student]password=[student]

Input
field=value1&field=value2&field=value3 http://example.com/over/there?name=ferret END

Output
field=[value1, value2, value3] name=[ferret]

Input
foo=%20foo&value=+val&foo+=5+%20+203 foo=poo%20&value=valley&dog=wow+ url=https://softuni.bg/trainings/coursesinstances/details/1070 https://softuni.bg/trainings.asp?trainer=nakov&course=oop&course=php END

Output
foo=[foo, 5 203]value=[val] foo=[poo]value=[valley]dog=[wow] url=[https://softuni.bg/trainings/coursesinstances/details/1070] trainer=[nakov]course=[oop, php]

Problem 10.* Use Your Chains, Buddy

This problem is from the JavaScript Basics Exam (9 January 2015). You may check your solution here.

You are in Cherny Vit now and there are 12km to Anchova Bichkiya Hut. You need to get there by car. But there is so much snow that you need to use car chains. In order to put them on the wheels correctly, you need to read the manual. But it is encrypted…

As input you will receive **an HTML document** as a **single string**. You need to get the text from **all the tags** and replace all characters which are **not small letters and numbers** with a space **" "**. After that you must decrypt the text – all letters **from a to m** are **converted** to letters **from n to z** (a → n; b → o; … m → z). The letters **from n to z** are **converted** to letters **from a to m** (n → a; o → b; … z → m). All **multiple** spaces should then be replaced by only **one space**.

For example, if we have **" Santa znahny # grkg ()&^^^&12 "** we extract **"znahny # grkg ()&^^^&12"**. Every **character** that is **not a small letter or a number** is replaced with a space (**"znahny grkg 12"**). We convert each small letter as described above (**"znahny grkg 12"** → **"manual text 12"**) and replace all multiple spaces with only **one space** (**"manual text 12"**). Finally, we concatenate the decrypted text from all ** ** tags (in this case, it's only one). And there you go – you have the manual ready to read!

Input

The input is read from the console and consists of just one line – the **string** with the **HTML document**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

Print on a single line the decrypted text of the manual. See the given **examples** below.

Constraints

- Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

Examples

Input
<html><head><title></title></head><body> hello **znahny!@#%&&&&**** <button>dsad</button> **grkg^^^^%%%)))([]12** </body></html>
Output
manual text 12
Input
<html><head><title></title></head><body> Intro   - Item01   - Item02   - Item03   **jura qevivat va jrg fyvccrel fabjl** <button>Click me, baby!</button> **pbaqvgvbaf fabj qpunvaf ner nofbyhgryl rffragvny sbe fnsr unaqyvat nygubhtu fabj punvaf znl ybbx** This manual is false, do not trust it! The illuminati wrote it down to trick you! **vagvzvqngvat gur onfvp vqrn vf ernyyl fvzcyr svg gurz bire lbhe gverf qevir sbejneq fybjyl naq gvtugra gurz hc va pbyq jrg** **pbaqvgvbaf guvf vf rnfvre fnvq guna qbar ohg vs lbh chg ba lbhe gverf** </body>
Output
when driving in wet slippery snowy conditions snow dchains are absolutely essential for safe handling although snow chains may look intimidating the basic idea is really simple fit them over your tires drive forward slowly and tighten them up in cold wet conditions this is easier said than done but if you put on your tires

Problem 11. ** Semantic HTML

This problem is originally from the PHP Basics Exam (31 August 2014). You may check your solution here.

You are given an **HTML code**, written in the old **non-semantic** style using tags like ** **, ** **, etc. Your task is to write a program that **converts this HTML to semantic HTML** by changing tags like ** ** to their semantic equivalent like **<header>**.

The non-semantic tags that should be converted are **always **s and have either **id** or **class** with one of the following values: "**main**", "**header**", "**nav**", "**article**", "**section**", "**aside**" or "**footer**". Their corresponding closing tags are always followed by a comment like **<!-- header -->**, **<!-- nav -->**, etc. staying at the same line, after the tag.

Input

The input will be read from the console. It will contain a variable number of lines and will end with the keyword "**END**".

Output

The output is the semantic version of the input HTML. In all converted tags you should **replace multiple spaces** (like **<header style="color:red">**) with a single space and remove excessive spaces at the end (like **<footer >**). See the examples.

Constraints

- Each line from the input holds either an HTML **opening tag** or an HTML **closing tag** or HTML **text content**.
- There will be no tags that span several lines and no lines that hold multiple tags.
- Attributes values will always be enclosed in **double quotes "**.
- Tags will never have **id** and **class** at the same time.
- The HTML will be **valid**. Opening and closing tags will match correctly.
- **Whitespace** may occur between attribute names, values and around comments (see the examples).
- Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output

`<div id="header">`
`</div> <!-- header -->`
`END`
	`<header>`
	`</header>`

`<div style="color:red" id="header">`
`</div> <!-- header -->`
`END`
	`<header style="color:red">`
	`</header>`

`<div class="header" style="color:blue">`
`</div> <!-- header -->`
`END`
	`<header style="color:blue">`
	`</header>`

`<div align="left" id="nav" style="color:blue">`
`  <ul class="header">`
`  <li id="main">`
`  Hi, I have id="main".`
`  </li>`
`  </ul>`
`</div> <!-- nav -->`
`END`
	`<nav align="left" style="color:blue">`
	`  <ul class="header">`
	`  <li id="main">`
	`  Hi, I have id="main".`
	`  </li>`
	`  </ul>`
	`</nav>`

`  <p class = "section" >`
`  <div style = "border: 1px" id = "header" >`
`  Header`
`  <div id = "nav" >`
	`  <p class = "section" >`
	`  <header style = "border: 1px">`
	`  Header`
	`  <nav>`

Nav
`</div>` `<!--` nav `-->`
`</div>` `<!--header-->`
`</p> <!-- end paragraph section -->`
END
	Nav
	`</nav>`
	`</header>`
	`</p> <!-- end paragraph section -->`