# More 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/443>.

## Serialize String

You have been tasked to serialize a string. The serialization is done in a special way, in which a character from that string is saved with the indexes at which it is found.

The input will consist of a single input line, containing a single string, which may consist of **ANY ASCII** character. Your task is to serialize the string in the following way:

{char}:{index1}/{index2}/{index3}

The char will be the **character**, and the indexes, will be the **indexes** it is **found** at in the **string**.

**Note:** This problem is a **string problem**, and should **ONLY** use **strings** in its **solution**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| abababa | a:0/2/4/6  b:1/3/5 |
| avjavamsdmcalsdm | a:0/3/5/11  v:1/4  j:2  m:6/9/15  s:7/13  d:8/14  c:10  l:12 |

## Stateless

You will be **given groups** of **2 strings**, each on a **new line**. There will **ALWAYS** be at **least 2 input lines**, and there will **NEVER** be a case when there **are less than 2 input strings**, for a **given element of the input**.

Now to the main logic – the **elements of the input**. You can **refer** to the elements of the input as states.

Each state also has a fiction – the collapsing factor. Your task is to **collapse** **each** **state**, by its **given** **fiction**.

The collapsing is done by **removing all occurrences** of the **fiction** in the **state**, and after that – **removing** the **first** and **last element** of the **fiction**. You must then **repeat the process**, until the **fiction’s length** becomes 0.

When you finish the process, you must **print what is left** from the **state**. If the state is also empty, you should print “(void)”. **NOTE**: **Border spaces** should be **removed**.

Both the **state** and the **fiction** are **strings**, and will be **given each** on a **separate line**. You must read **sequences** of **DOUBLE lines**, and **print** the **result** from the **collapsing**, until you receive the command “collapse”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| astalavista baby  aaa  aaaa  aa  this will be funny rhight  this  collapse | stlvist bby  (void)  will be funny rght |
| bow chicka mow wow  mow  ahaia  hai  collapse | bw chicka ww  (void) |

## Pyramidic

You will be **given N** – an **integer**. On the next **N input lines**, you will be given **N strings**, which may consist of **any ASCII character**.

Your task is to find the **BIGGEST** **pyramid formation** of **occurrences** of a **SINGLE CHARACTER**, **throughout** **the strings**.

The pyramid is formed by **finding** a **character** on a line, then **finding** **3 consecutive** (**next to each, other**) occurrences of the **same character** on the **next line**, then finding **5 consecutive** occurrences on the next line and so on. . .

Example:  
abacd  
bbbcd  
bbbbb

Result:

b  
bbb  
bbbbb

Check the examples for more info.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  asdfghjkl  asdgggjkl  asgggggkl  agggggggl  ggggggggg | g  ggg  ggggg  ggggggg  ggggggggg |
| 7  abcdefg  aaadc\\  cbaaaaa  d  ddddasd  !!ddddd!!!!!!!!...  dddddddd | d  ddd  ddddd  ddddddd |

## Nilapdromes \*

Nilapdromes are similar to palindromes, but are quite different. **Nilapdromes** are words which have 1 substring of random characters in the middle, called – the **core**, and **2 identical substrings**, surrounding it, called – the **borders**.

Examples of **nilapdromes** are: “aba”, “asdthisasd”, “baumyaubau”. . .

Examples of **INCORRECT** **nilapdromes** are: “abbc”, “SDSD”, “\_,#$x$#,\_y”.

For example, the **nilapdrome** “baumyaubau” – the **core** is “myau” and the **borders** are “bau”.

You will be receiving input lines, containing **exactly one** nilapdrome, each, until you receive the command “end”.

Your task is to **make**, **from each nilapdrome** – a **new nilapdrome**, with **borders** – **equal** to the **core** (**middle substring**) of the **given one**, and **core** – **equal** to the **borders** of the **given one**.

For example, the **nilapdrome** “baumyaubau” should **result** in “myaubaumyau”.

You should **print** **each result nilapdrome**, after you’ve created it, **BEFORE** reading the **next one.**

**INCORRECT** **nilapdromes**, should be **IGNORED**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| aba  asdthisasd  baumyaubau  end | bab  thisasdthis  myaubaumyau |
| everythingnothingeverything  invalid  donenodedonee  abbc  sdsd  ssdd  dssd  end | nothingeverythingnothing  ssdss |

\b([a-zA-Z]+)([a-zA-Z]+)\1\b