6536 Beautiful Soup

Coach Pang has a lot of hobbies. One of them is playing with "tag soup" with the help of Beautiful Soup. Coach Pang is satisfied with Beautiful Soup in every respect, except the `prettify()` method, which attempts to turn a soup into a nicely formatted string. He decides to rewrite the method to prettify a HTML document according to his personal preference. But Coach Pang is always very busy, so he gives this task to you. Considering that you do not know anything about "tag soup" or Beautiful Soup, Coach Pang kindly left some information with you:

In Web development, "tag soup" refers to formatted markup written for a web page that is very much like HTML but does not consist of correct HTML syntax and document structure. In short, "tag soup" refers to messy HTML code.

Beautiful Soup is a library for parsing HTML documents (including "tag soup"). It parses "tag soup" into regular HTML documents, and creates parse trees for the parsed pages.

The parsed HTML documents obey the rules below.

HTML

- HTML stands for HyperText Markup Language.
- HTML is a markup language.
- A markup language is a set of markup tags.
- The tags describe document content.
- HTML documents consist of tags and texts.

Tags

- HTML is using *tags* for its syntax.
- A tag is composed with special characters: '<', '>' and '/'.
- Tags usually come in pairs, the *opening tag* and the *closing tag*.
- The *opening tag* starts with '<' and the *tagname*. It usually ends with a '>'.
- The *closing tag* starts with '</' and the same tagname as the corresponding opening tag. It ends with a '>'.
- There will not be any other angle brackets in the documents.
- Tagnames are strings containing only lowercase letters.
- Tags will contain no line break ('\n').
- Except tags, anything occured in the document is considered as *text content*.

Elements

- An element is everything from an opening tag to the matching closing tag (including the two tags).

- The element content is everything between the opening and the closing tag.
- Some elements may have no content. They're called *empty elements*, like `<hr></hr>`.
- Empty elements can be closed in the opening tag, ending with a '`/>`' instead of '`>`'.
- All elements are closed either with a closing tag or in the opening tag.
- Elements can have attributes.
- Elements can be nested (can contain other elements).
- The `<html>` element is the container for all other elements, it will not have any attributes.

Attributes

- Attributes provide additional information about an element.
- Attributes are always specified in the opening tag after the tagname.
- Tag name and attributes are separated by single space.
- An element may have several attributes.
- Attributes come in `name="value"` pairs like `class="icpc"`.
- There will not be any space around the '`=`'.
- All attribute names are in lowercase.

A Simple Example `<a href="http://icpc.baylor.edu/">ACM-ICPC</a>`

- The `<a>` element defines an HTML link with the `<a>` tag.
- The link address is specified in the href attribute.
- The content of the element is the text "ACM-ICPC"

You are feeling dizzy after reading all these, when Coach Pang shows up again. He starts to spout for hours about his personal preference and you catch his main points with difficulty. Coach Pang says:

> Your task is to write a program that will turn parsed HTML documents into formatted parse trees. You should print each tag or text content on its own line preceded by a number of spaces that indicate its depth in the parse tree. The depth of the root of the a parse tree (the `<html>` tag) is 0. He is satisfied with the tags, so you **shouldn't** change anything of any tag. For text content, throw away unnecessary white spaces including space (ASCII code 32), tab (ASCII code 9) and newline (ASCII code 10), so that words (sequence of characters without white spaces) are separated by single space. There should not be **any trailing space** after each line nor **any blank line** in the output. The line contains only white spaces is also considered as blank line. You quickly realize that **your only job is to deal with the white spaces**.

Input

The first line of the input is an integer T representing the number of test cases.

Each test case is a valid HTML document starts with a '`<html>`' tag and ends with a '`</html>`' tag. See sample below for clarification of the input format.

The size of the input file will not exceed 20KB.

Output

For each test case, first output a line '`Case #`x`:`', where x is the case number (starting from 1).

Then you should write to the output the formatted parse trees as described above. See sample below for clarification of the output format.

Hint: Please be careful of the number of leading spaces of each line in above sample output.

Sample Input

```
2
<html><body>
<h1>ACM
ICPC</h1>
<p>Hello<br/>World</p>
</body></html>
<html><body><p>
Asia Chengdu Regional</p>
<p class="icpc">
ACM-ICPC</p></body></html>
```

Sample Output

```
Case #1:
<html>
 <body>
  <h1>
   ACM ICPC
  </h1>
  <p>
   Hello
   <br/>
   World
  </p>
 </body>
</html>
Case #2:
<html>
 <body>
  <p>
   Asia Chengdu Regional
  </p>
  <p class="icpc">
   ACM-ICPC
  </p>
 </body>
</html>
```