The goal of the task is to write a simple CSS processing engine. As part of the task, you need to read CSS sections interleaved with command sections from standard input. CSS sections should be parsed and placed in the appropriate structures, and command sections should be parsed and executed, printing any results to standard output (after ==).

CSS.

• Processing begins with the reading of CSS declarations. CSS is syntactically correct and consists of blocks of attributes possibly preceded by selectors. The absence of selectors is legal (it would mean attributes applied to everything).

• Selectors are separated by commas. Legal selectors for CSS are allowed, but it can be assumed that they do not contain commas or curly braces.

• The block of attributes is enclosed in curly braces.

• Attributes are separated by semicolons and consist of a name (property) and a value separated by a colon. A semicolon may or may not follow the last attribute in the block.

• Legal CSS constructions can be used as attribute values, but for simplicity, it can be safely assumed that any strings are not malicious, i.e. do not contain escaped quotation marks, curly braces, or semicolons.

• If a specific attribute name appears multiple times in a block, treat it as a single occurrence, with the last value being significant.

• Both selectors, attribute names, and attribute values do not require semantic interpretation, i.e. they are treated as values after trimming whitespace. For example, 'margin-left: 8px' and 'margin: 4px 7px 4px 7px' are treated as separate, unrelated attributes with the names 'margin-left' and 'margin' and the values '8px' and '4px 7px 4px 7px', respectively. Similarly, selectors are treated as values and do not require interpretation, i.e. 'h1' and 'h1.theme' are treated as separate, unrelated selectors.

• Simplified CSS does not contain comments or @-type selectors, and blocks cannot be nested.

• For the purposes of all tests, it can be assumed that no selector or attribute is split across multiple lines (multiple separators and/or attributes can appear in one line).

Commands.

In the following commands, i and j are positive integers (of int type), while n is a legal attribute name.

???? - start of command section;

**** - resume reading CSS;

? - print the number of CSS sections;

i,S,? - print the number of selectors for section number i (section and attribute numbers start from 1), if there is no such block, skip;

i,A,? - print the number of attributes for section number i, if there is no such block or section, skip;

i,S,j - print the j-th selector for the i-th block (section and attribute numbers start from 1), if there is no section or selector, skip;

i,A,n - print the value of the attribute with the name n for the i-th section, if there is no such attribute, skip;

n,A,? - print the total (for all blocks) number of occurrences of attribute named n (duplicates should be removed when reading). It can be 0;

z,S,? - print the total (for all blocks) number of occurrences of selector z. It can be 0;

z,E,n - print the value of the attribute named n for the selector z, in case of multiple occurrences of selector z, take the last one. If there is no such attribute, skip;

i,D,* - remove the entire section number i (i.e., separators+attributes), after successful execution, print "deleted";

i,D,n - remove the attribute named n from the i-th section, if the section becomes empty as a result of the operation, it should also be removed (along with any selectors), after successful execution, print "deleted".

Implementation notes:

Selectors and attributes should be stored as lists. Individual CSS sections should be stored in a doubly linked list (to efficiently perform the E command). To better utilize memory, the list should include an array T=8 of structures representing a block (where T is a constant that can be changed at compile time) and a counter for currently occupied structures (due to potential element deletions). Counters should be utilized to speed up operations parameterized by the cell number, i. When allocating a new node, a T-sized array should be created. When adding elements, if there is free space in the list node, it should be used before allocating new nodes. If an empty array remains after removing elements, the node should be removed. It is not necessary to move elements between nodes, merge nodes, etc.

Example:

```
#breadcrumb
{
      width: 80%;
      font-size: 9pt;
}

h1, body {
      min-width: 780px;
      margin: 0;
      padding: 0;
      font-family: "Trebuchet MS", "Lucida Grande", Arial;
      font-size: 85%;
      color: #666666;
}

h1, h2, h3, h4, h5, h6 {color: #0066CB;}
????
?
1,S,?
1,S,1
1,A,?
2,A,font-family
h1,S,?
color,A,?
h1,E,color
1,A,padding
1,D,*
?
2,D,color
?

****
h1, h2, h3, h4, h5, h6 {color: #0066FF;}
????
?
```

Result:

```
? == 3
1,S,? == 1
1,S,1 == #breadcrumb
1,A,? == 2
2,A,font-family == "Trebuchet MS", "Lucida Grande", Arial
h1,S,? == 2
color,A,? == 2
h1,E,color == #0066CB
1,D,* == deleted
? == 2
2,D, color == deleted
? == 1
? == 2
```