Problem 6: Making Mountains Out Of Molehills

A macro processor is a symbol processing program. It takes a stream of characters as its input, and produces a stream of characters as its output. A "macro" is similar to ordinary function in that it has a definition, and when invoked, that definition is applied to the argument(s) to yield the result. In this problem, you will develop a simple macro processor.

A "macro call" consists of a name and a list of arguments, separated by commas. The name is preceded by '[' (a left bracket) and the last argument is followed by ']' (a right bracket). For example, "[doit,to,it]" calls the macro named "doit" with two arguments, "to" and "it". "[random]" calls the macro named "random" with no arguments.

Before a macro can be called, it must be defined by associating its name with a symbol string. This definition string may contain the special constructions "$1" through "$9" to reference the first through the ninth macro parameters. "$0" references the macro's name itself. When the macro is called, these constructions are literally replaced by the values of the parameters. For example, suppose the definition string for the macro named "321" was "$3-$2-$1". The macro call "[321,This,is,fun]" would yield the output "fun-is-This". A macro call can appear anywhere. For example, the macro call "[321,[321,A,B,C],D,E]" would yield the output "E-D-C-B-A".

Input enclosed in '<' and '>' prevents the evaluation of the text enclosed, allowing special characters like '[', ']', ',' and '$' to be used in other than their usual contexts. Thus the macro call "[321,<$>,<[>,<,>]" would yield ",-[-$".

Macros are defined using the predefined macro named "def", which has two arguments. The first argument is the name of the macro being defined, and the second argument is the defining symbol string for the macro. The "321" macro definition is "[def,321,<$3-$2-$1>]". Note that the definition is enclosed in '<' and '>' to prevent "$1", "$2", and "$3" from being interpreted as parameter references to def. The def macro produces no output. Naturally, the def macro isn't defined using def, but is treated specially by the implementation.

Processing
The input stream is processed character by character and copied to the output until a macro call is encountered, or the input is exhausted (which terminates processing). A macro call is evaluated as described below, with the result (if any) copied to the output.

1. The macro name and the parameters are evaluated in sequence from left to right. This may require evaluating additional macro calls, which must be processed recursively.

2. When the argument list is complete (that is, when the closing ']' is encountered) the definition of the macro being called is scanned in the same manner as the original input stream except that occurrences of "$0", "$1", and so forth are replaced literally by the corresponding arguments. The result of the macro call is the symbol stream produced by this scan.

3. When the macro call is completed, the macro name and the arguments are discarded, and processing resumes at the point where it was interrupted by the macro call.

Limits and Caveats
Macro names and arguments will contain no more than 32 characters each. The defining string for a macro will contain no more than 100 characters. Macros will never be defined more than once (that is, the same macro name will not be used more than once as the first argument to "def"). Macro calls will always provide the correct number of arguments. Character case is significant in comparisons. All input characters, including end of line characters, are to be processed through the macro processor. No output line will contain more than 80 characters, including the end of line character. The input is guaranteed to be correct.

Input

There will be multiple cases to consider. The input for each case begins with a line containing a single integer between 1 and 10, that specifies the number of lines of text immediately following that will be used as input to the macro processor. None of these lines will contain more than 80 characters, so the input to the macro processor will contain at most 810 characters. The last case will be followed by a line containing the integer 0.

Output

For each input case, display the case number (1, 2, …), a line containing 79 hyphens, the output from the macro processor, another line containing 79 hyphens, and a blank line.

In the sample input shown below, assume that the last visible character on each line is immediately followed by the end of line character. Blank lines in the expected output are shown here as **BLANK** for clarity, but these should actually be totally blank in your output.

Sample Input

```
1
This is just copied (including end of line).
1
[def,321,<$3-$2-$1>][321,This,is,fun]
1
[def,321,<$3-$2-$1>][321,[321,A,B,C],D,E]
1
[def,321,<$3-$2-$1>][321,This,is,fun][321,[321,A,B,C],D,E]
3
[def,321,<$3-$2-$1>]
[321,This,is,fun]
[321,[321,A,B,C],D,E]
3
[def,A,<$1[B]$2>]
[def,B,*B*]
[A,1,2]
2
[def,#,<[-]$1[-]$2[-]$3[-]>][def,-,<$0>][def,DEF,def][[DEF],X,THIS IS X]
[X][#,DEF,#,X]
0
```

Expected Output

```
Case 1
-------------------------------------------------------------------------------
This is just copied (including end of line).
-------------------------------------------------------------------------------
**BLANK**
Case 2
-------------------------------------------------------------------------------
fun-is-This
-------------------------------------------------------------------------------
**BLANK**
Case 3
-------------------------------------------------------------------------------
E-D-C-B-A
-------------------------------------------------------------------------------
**BLANK**
Case 4
-------------------------------------------------------------------------------
```

```
fun-is-ThisE-D-C-B-A
--------------------------------------------------------------------------------
**BLANK**
Case 5
--------------------------------------------------------------------------------
**BLANK**
fun-is-This
E-D-C-B-A
--------------------------------------------------------------------------------
**BLANK**
Case 6
--------------------------------------------------------------------------------
**BLANK**
**BLANK**
1*B*2
--------------------------------------------------------------------------------
**BLANK**
Case 7
--------------------------------------------------------------------------------
**BLANK**
THIS IS X-DEF-#-X-
--------------------------------------------------------------------------------
**BLANK**
```