4150 Filtration

Digital Signal Processing is used for such clever effects as the "echo" often heard in music, or that annoying modulation done with the voices of certain singers. The field makes the use of *Finite Impulse Response* (FIR) filters to make these effects happen.

Consider an input stream of samples, represented as a series of integers between 0 and 255 inclusive. (This is, you should not be surprised to discover, the range of an *8-bit* sample.) These samples come in a very specific sequence, one after the other, as the digital representation of a given waveform. A FIR mixer will take multiple streams and combine them into one, a FIR echo filter will take one input and provide a single output, and so on.

A FIR filter can be represented as an equation, with the input signals on one side and the output on the other. For example:

```
Y = X[0] + X[-5]
```

represents an echo filter (more specifically, a post-echo filter); a given sample of Y is equal to the value of X at the same location in the input stream, plus the value of the sample that occurred five samples prior in X. Values that are not available (such as $X[-5]$ for the first four samples of X) are set to 0.

In this problem, all FIR equations obey the following Backus-Naur Form (BNF) grammar:

EQUATION	::=	***STREAM*** _ "=" _ ***EXPR***
STREAM	::=	A single upper-case letter representing a sample stream, such as `A` or `X`
EXPR	::=	***VALUE*** | ***SAMPLE*** | ***EXPR*** _ ***OPER*** _ ***EXPR*** | "(" _ ***EXPR*** _ ")"
VALUE	::=	A floating-point number, such as `0.25`, `5`, or `-1.5`
SAMPLE	::=	***STREAM*** "[" ***OFFSET*** "]"
OFFSET	::=	An integer representing the sample offset into the stream, such as `0`, `1`, or `-5`, between `-100` and `100` inclusive.
OPER	::=	"*" | "+" | "-"

Operations are handled in the standard order of precedence (parentheses first, multiplication before addition and subtraction, otherwise left-to-right), and the resulting value is rounded *down* to the nearest integer between 0 and 255 only *after* all calculations (if any) are done by the FIR filter. The _ symbol in the above grammar specifies a series of one or more spaces. Note that whitespace in an equation is only allowed where explicitly specified by the above grammar.

For example, a simple low-pass filter could be expressed as:

```
Z = 0.5 * Y[0] + 0.25 * Y[-1] + 0.25 * Y[1]
```

Each output value of Z is based on the matching value in Y, modified by the nearest values of Y. A simple mixer can be represented as follows:

```
D = C[0] + B[0]
```

although the clipping problems that such a filter would have should be apparent.

Obviously, for complicated effects, a number of filters can be connected together. In this problem, this is represented by **STREAM**; any relevant stream will either be an input value (the source audio, for example) or the output of a single FIR filter. However, a stream may be *used* as an input by more than one other filter. This constitutes a *filter network*.

Given a series of definitions of FIR filters and starting inputs, your task is to provide all of the outputs of the various filters. No FIR filter will have more than 10 operators or more than ten pairs of parentheses, nor will its representation use more than 80 characters. There will be at least one input stream and at least one output stream per data set, and all streams referenced in a filter equation will be defined in the same data set. All input streams and output streams will have the same number of samples, and there will be no ”feedback loops” described by a filter network; that is, no filter will have input dependent on its output.

Although they sure sound cool when Pete Townshend uses them.

Input

Input to this problem will begin with a line containing a single integer D ($1 \leq D \leq 100$) indicating the number of data sets. Each data set consists of the following components:

- A line containing a single integer N ($2 \leq N \leq 26$) indicating the number of streams in the data set;
- A line containing a single integer S ($1 \leq S \leq 100$) indicating the number of samples in every stream of the data set;
- A series of N lines, each representing one of the streams. There will be no duplicate stream names in a data set, and each stream is one of either:
  - an input stream, in which case its representation is of the form ‘$STREAM$ `%` $sam1$ $sam2$ $sam3$ … $samS$’, where $STREAM$ is a single capital letter, and $sam1$ is the first sample of the input stream, $sam2$ is the second sample, and so on. The individual samples and the ‘`%`’ operator are all separated with whitespace.
  - a FIR filter, in which case its representation is of the form ‘$STREAM$ `=` $EXPR$’, as described above.

Output

For each data set in the input, output the heading ‘`DATA SET #`k’ where k is 1 for the first data set, 2 for the second, and so on. Then print the output sample streams for every FIR filter in the data set, in alphabetical order, in the format ‘$STREAM$ `%` $sam1$ $sam2$ $sam3$ … $samS$’.

Sample Input

```
3
3
5
A % 10 20 30 20 10
B = A[0] + A[-1]
C = 0.5 * ( A[0] + B[0] )
3
5
Z = 0.5 * Y[0] + 0.5 * B[0]
Y % 50 10 50 10 50
B % 10 50 10 50 15
3
5
A % 1 2 3 4 5
B = A[-2]
C = B[2]
```

Sample Output

```
DATA SET #1
B % 10 30 50 50 30
C % 10 25 40 35 20
DATA SET #2
Z % 30 30 30 30 32
DATA SET #3
B % 0 0 1 2 3
C % 1 2 3 0 0
```