## 586 Instant Complexity

Analyzing the run-time complexity of algorithms is an important tool for designing efficient programs that solve a problem. An algorithm that runs in linear time is usually much faster than an algorithm that takes quadratic time for the same task, and thus should be preferred.

Generally, one determines the run-time of an algorithm in relation to the 'size' $n$ of the input, which could be the number of objects to be sorted, the number of points in a given polygon, and so on. Since determining a formula dependent on $n$ for the run-time of an algorithm is no easy task, it would be great if this could be automated. Unfortunately, this is not possible in general, but in this problem we will consider programs of a very simple nature, for which it is possible. Our programs are built according to the following rules (given in BNF), where $< number >$ can be any non-negative integer:

- $< Program >$ ::= "`BEGIN`" $< Statementlist >$ "`END`"
- $< Statementlist >$ ::= $< Statement >$ | $< Statement >$ $< Statementlist >$
- $< Statement >$ ::= $< LOOP - Statement >$ | $< OP - Statement >$
- $< LOOP - Statement >$ ::= $< LOOP - Header >< Statementlist >$ "`END`"
- $< LOOP - Header >$ ::= "`LOOP`" $< number >$ | "`LOOP n`"
- $< OP - Statement >$ ::= "`OP`" $< number >$

The run-time of such a program can be computed as follows: the execution of an OP-statement costs as many time-units as its parameter specifies. The statement list enclosed by a LOOP-statement is executed as many times as the parameter of the statement indicates, i.e., the given constant number of times, if a number is given, and $n$ times, if `n` is given. The run-time of a statement list is the sum of the times of its constituent parts. The total run-time therefore generally depends on $n$.

### Input

The input file starts with a line containing the number $k$ of programs in the input. Following this are $k$ programs which are constructed according to the grammar given above. Whitespace and newlines can appear anywhere in a program, but not within the keywords `BEGIN`, `END`, `LOOP` and `OP` or in an integer value. The nesting depth of the LOOP-operators will be at most 10.

### Output

For each program in the input, first output the number of the program, as shown in the sample output. Then output the run-time of the program in terms of $n$; this will be a polynomial of degree $\leq 10$. Print the polynomial in the usual way, i.e., collect all terms, and print it in the form "`Runtime = `$a$`*n^10+`$b$`*n^9+ . . . +`$i$`*n^2+ `$j$`*n+k`", where terms with zero coefficients are left out, and factors of 1 are not written. If the runtime is zero, just print "`Runtime = 0`".

Output a blank line after each test case.

## Sample Input

```
2
BEGIN
LOOP n
OP 4
LOOP 3
LOOP n
OP 1
END
OP 2
END
OP 1
END
OP 17
END
BEGIN
OP 1997 LOOP n LOOP n OP 1 END END
END
```

## Sample Output

```
Program #1
Runtime = 3*n^2+11*n+17

Program #2
Runtime = n^2+1997
```