Preamble

Please note the following very important details:

1) Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin` or equivalent. Input will be redirected from a file to form the input to your submission.

2) Write all output to the screen, i.e. use `stdout`, `System.out`, `cout` or equivalent. Do not write to `stderr`. Do NOT use, or even include, any module that allows direct manipulation of the screen, such as `conio`, `Crt` or anything similar. Output from your program is redirected to a file for later checking. Use of direct I/O means that such output is not redirected and hence cannot be checked. This could mean that a correct program is rejected!

3) Unless otherwise stated, all *integers* will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by white space, i.e. spaces or tabs.

4) Unless otherwise stated, a *word* is a continuous sequence of lower case letters without any punctuation or other characters and, in particular, without intervening white space. As with numbers, successive words will be separated by white space.

6) Unless otherwise stated, a *name* is a continuous sequence of letters without any punctuation or other characters and, in particular, without intervening white space. As with words, successive names will be separated by white space.

7) Unless otherwise stated, a *string* is a continuous sequence of characters without any intervening white space. As with words, successive strings will be separated by white space.

8) If it is stated that 'a line contains no more than *n* characters', this does not include the character(s) specifying the end of line.

Program Names

Still to be determined

Problem A One, two, many ... 10 points

English, in common with many other languages, distinguishes between singular and plural forms of nouns (and verbs too, but that is a different problem -- maybe next year). The general rule is that a simple word takes an ‘s’ in the plural, thus we talk of ‘one dog’ but ‘many dogs’ and so on, but there are many variations.

Given the following rules, write a program that will pluralise a singular word or singularise a plural word. The rules are given from the most specific to the most general, you should consider them in that order and apply the first rule that matches. The rules are given in the form *string1 –> *string2 (example), which means that any singular word that ends in string1 is pluralised by replacing string1 with string2 and *vice versa* for singularising. If you do not find a rule that applies for going from plural to singular, the leave the word unchanged ('many sheep' -> 'one sheep').

> *ouse –> *ice (mouse -> mice)
> *us –> *i (abacus -> abaci)
> *um –> *a (medium -> media)
> *on –> *a (criterion -> criteria)
> *x –> *xes (tax -> taxes)
> *y –> *ies (fancy -> fancies)
> *s –> *ses (bus -> buses)
> *sh –> *shes (bush -> bushes)
> * –> *s (dog -> dogs, horse -> horses)

Input will consist of a series of lines terminated by a line containing a single ‘#’. Each line will start with either the word ‘one’ or the word ‘many’ followed by a space and a singular or plural word as appropriate, where a word is as defined in the preamble and will consist of no more than 20 letters.

There will be one line of output for every line of input, except the last. Each output line will contain a copy of the input line, the characters `' -> '` and the ‘opposite’ of the input, in that order. Follow the format used in the sample output section.

Sample input

```
one mouse
many buses
one bush
many dogs
many sheep
one sheep
#
```

Sample output

```
one mouse -> many mice
many buses -> one bus
one bush -> many bushes
many dogs -> one dog
many sheep -> one sheep
one sheep -> many sheeps
```

Problem B Digital Madness 10 points

In this problem you will be given a sequence of operations corresponding to the digits from 0 to 9, and asked to compute the value of the resulting expression when each digit of a number except the last is replaced by that digit and its operation. For example, the sequence ++++++++++ would simply result in converting a number to the sum of its digits. Similarly, if the operation sequence is +–*+–*+–*+ then the number 34563456 becomes 3 + 4 – 5 * 6 + 3 + 4 – 5 * 6 = 84.

Notice that the normal precedence rules do not apply — evaluation proceeds left to right. You can assume that all results and all intermediate values will fit into a standard 32-bit signed integer.

Input will consist of multiple problem instances. Each problem instance starts with a line containing an operation sequence (a sequence of 10 characters, each one of +, -, *), followed by one or more lines of numbers to be evaluated (each with between 1 and 20 digits). The list of numbers will be terminated by the single digit 0; the sequence of problem instances will be terminated by a line consisting of a single #.

Output should be of the form shown below. Each digit except the last is followed by a space, its associated operator, and another space. The last digit is followed by a space, an = sign, another space, and then the result of evaluating the expression. Separate the output for successive problem instances by a blank line.

Sample input

```
++++++++++
2345
0
+-*+-*---+
123
2340
9876543210
0
#
```

Sample output

```
2 + 3 + 4 + 5 = 14

1 - 2 * 3 = -3
2 * 3 + 4 - 0 = 10
9 + 8 - 7 - 6 - 5 * 4 - 3 + 2 * 1 - 0 = -5
```

Problem C Independence Day 10 points

In one of the many current liberation wars, the people of Southeastern Southwestland are fighting a bitter struggle for independence. They have all the normal trappings of a modern state — a democratically elected president (well almost - there were two candidates and they each received half the votes so they share the office) and a representative Parliament (both the voters are sitting members), however the population of 3.5 (the Minister of Finance spends half his time at his day job in Southwestern Southeastland while the Minister of Agriculture looks after the farm) is unable to afford flags to arouse patriotic fervour in support of their struggle. This is where you come in.

Write a program that will produce flags for this wannabe country in any size and background that they wish. Size 1 and 2 flags looks like this:

```
    +-+-+   +--+--+
    |\|/|   |\ | /|
    +-+-+   | \|/ |
    |/|\|   +--+--+
    +-+-+   | /|\ |
            |/ | \|
            +--+--+
```

Input will consist of a series of lines each containing a single 'filler' character (possibly a space), a space and an integer *n* ($1 \le n \le 30$) terminated by a line containing a hash and a zero (# 0). This line should not be processed.

Output should be of the form shown above, with the spaces replaced by the filler character given. Note that *n* refers to the 'size' of the flag, i.e. the number of '/' or '\' characters in a quadrant. Follow the format shown in the Sample Output below. Separate successive flags by a blank line.

Sample input

```
  1
B 2
# 0
```

Sample output

```
+-+-+
|\|/|
+-+-+
|/|\|
+-+-+

+--+--+
|\B|B/|
|B\|/B|
+--+--+
|B/|\B|
|/B|B\|
+--+--+
```

Problem D Scheduling 10 points

Regular background tasks run to some sort of schedule. In this problem each daily schedule is represented as a list of sub-schedules. Each sub-schedule consists of three integers:

- the time of the first event in the sub-schedule (HHMM, $0 \le HH \le 23$; $0 \le MM \le 59$),
- the gap in minutes until the next event in the sub-schedule (1..1439),
- the number of events generated by this sub-schedule (1..1439).

So, for the sub-schedule 0100,60,3 there are three events, that occur at 0100, 0200 and 0300. Note that under the rules above it is possible that a sub-schedule will produce event times ≥ 1440 (which fall outside the number of minutes in a day). Such event times are to be ignored.

Write a program that, for a given current time and a particular event schedule, will compute the time at which the next event occurs. Note that the next event may not be until the next day.

The input consists of a number of schedules. Each schedule consists of an integer *n* on a line by itself, *n* lines each containing a sub-schedule, and a list of times of day. The sequence of schedules is terminated by a line containing a single zero (0). Each sub-schedule consists of a time, a gap in minutes, and the number of events, as described above. The list of times will be terminated by a line containing 2400.

Output for each data set starts with a line containing the word 'Schedule' followed by the data set number. This is followed by a series of lines, one for each specified time in the input, giving the time of the next scheduled event (which might be on the next day). Separate successive schedules by a blank line.

Sample input

```
2
0030 60 18
2000 60 1439
0029
0030
0031
2301
2400
2
0620 30 100
0715 30 60
2346
0000
1151
2400
0
```

Sample output

```
Schedule 1
0030
0030
0130
0030

Schedule 2
0620
0620
1215
```

Problem F Mapmaker's Mayhem 30 points

A large scale map of Gridonia represents the country as a rectangular grid. Each cell in the grid is marked either as land or water. The king of Gridonia wishes to know the areas of all the connected regions of land or water in his domain, where a connected region is one whose squares are all of the same type, and whose constituent squares meet along north-south or west-east edges, i.e. not just corner to corner. Thus in the map:

```
LLWLLL
LWLWLL
```

there are three regions of land (containing three, five, and one grid squares respectively) and three regions of water each containing one grid square. Regions are identified by the grid coordinates of the first square of the area encountered when scanning west to east in each row working north to south, i.e. the most westerly point of the northernmost row. Grid coordinates start at (0,0) in the northwest corner, increasing in unit steps in the eastward and southward directions, thus the coordinates of the isolated square of land in the second row above would be (2 1).

Input consists of a series of maps. Each map starts with the size of the map given as a pair of integers *w h*, ($1 \leq w\ h \leq 50$) on a line by themselves, followed by a sequence of *h* lines each containing *w* 'L' or 'W' characters. The sequence of maps is terminated by an 'empty' map, i.e. a line containing a pair of zeroes (0 0).

Each map in the input produces several lines of output. The first line of each map's output identifies the map by a line with the word 'Map' followed by a space and the map identifier, a running number starting at 1. This is followed by as many lines as necessary identifying each region encountered by its starting coordinate as specified above and stating its area and type: 'land' or 'water'). Regions should be listed in the order encountered when scanning west to east in each row working north to south. Separate the output for successive maps by a blank line.

Sample input

```
6 2
LLWLLL
LWLWLL
5 5
LLLLL
LLLLL
LLLLL
LLWLL
LWLWL
0 0
```

Sample output

```
Map 1
0 0 3 land
2 0 1 water
3 0 5 land
1 1 1 water
2 1 1 land
3 1 1 water

Map 2
0 0 21 land
2 3 1 water
1 4 1 water
2 4 1 land
3 4 1 water
```

Problem G Map Overlaying 30 points

Dr Dosomething lives in Gridonia where cities were laid out as grids of numbered streets running North-South and avenues running East-West. The northernmost avenue and the west most street are both labelled 0 and successive streets and avenues are numbered with consecutive integers. However, the blocks were too large and subsequently a maze of alleys, side streets and arcades have infiltrated most of them. The largest city in the country has 100,000 streets and avenues.

Dr Dosomething is convinced that the causes of crime are simple and easily eradicated. He is convinced that crime is concentrated in areas where the street surfaces are poorly maintained, however the City Council are dubious. He has collected statistics from various sources and now has maps specifying the number of potholes in the pavements per block and also the number of crimes per block in the last year. When you examine the data you see that he has often consolidated the data from several blocks into rectangular areas and produced an average number of potholes and crimes accordingly. However he did this at different times so the rectangles do not coincide. He has asked you to determine all the intersecting areas between the two data sets and report the number of potholes and the number of crimes per block in such areas. You have grave doubts about the validity of the results, but the pay is good, so you accept the challenge.

Input will consist of a number of city maps. Each city map will consist of a pair of maps, where each map is a collection of non-overlapping rectangles together with an attribute — an integer value representing either the number of potholes or the number of crimes per block. A rectangle is specified by the coordinates of the northwest and southeast corners (as street, avenue pairs) followed by the number of potholes or crimes per block, i.e. by 5 integers. Note that the streets and avenues are well lit and well maintained so it is presumed that neither potholes nor crimes occur on them. Note also that data are not necessarily available for the entire city, so some areas may not be specified. The list of rectangles in each map will be terminated by 5 zeroes (0 0 0 0 0). The sequence of city maps will also be terminated by a line containing 5 zeroes.

Output will consist of a line identifying the scenario, consisting of the word 'Scenario' followed by a space and the number of the scenario — a running number starting at 1. This will be followed by the rectangles in the same format as above except that you will specify the values of both attributes. Each pair of intersecting rectangles in the input is to determine one rectangle in the output; output rectangles are not to be merged even if they could be. The output is to be sorted in lexicographic order, and the coordinate values are to be written in 6 columns with leading spaces, not leading zeros.

Sample input

```
0 0 100 100 101
0 0 0 0 0
50 0 150 50 201
50 50 150 150 202
0 0 0 0 0
0 0 0 0 0
```

Sample output

```
Scenario 1
    50     0   100    50   101   201
    50    50   100   100   101   202
```

Problem H Conversion to XML 30 points

XML has become a very popular data format in recent years. To solve this problem, you need to write a program that will convert from one particular non-XML format, to something close to XML.

The input data format is a number of name=value pairs, 1 per line. Each name consists of zero or more (id, sequence number) pairs, and a final id. The value is stored in an XML element whose name is the final id (note values appear in the same order in the XML output as they do in the input). The (preceding id, sequence number) pairs determine the nesting of that element within other XML elements. These rules for constructing names allow for grouping and repetition.

Input appears in a number of sets, each terminated by a line containing a #. The end of input is indicated by an empty set. Each name=value pair gives rise to an element of the form <finalid>value</finalid>. Also, any leading ids and sequence numbers determine the nesting of elements. Grouping is done based on matching both parts of an id, sequence number pair, as per the example below. Each id within a name is alphabetic and each sequence number is an integer greater than 0. Each input line is no more than 80 characters. Values do not contain new lines. Each value can contain any printable character except =, newline and those special to XML: < (less than), > (greater than) and & (ampersand).

Each input set should be output as an XML document within a Result element. An indent of 2 spaces should be used to highlight nesting, as shown in the sample output. A blank line should follow each XML document.

Sample input

```
Restaurant=Ernies
Menu1Name=Breakfast
Menu1Item1Food=Coffee
Menu1Item1Price=$2.00
Menu1Item2Food=Donuts
Menu1Item2Price=$3.00
Menu2Name=Lunch
Menu2Item1Food=Tea
Menu2Item1Price=$2.00
Menu2Item2Food=Soup
Menu2Item2Price=$4.00
Menu2Item3Food=Fish and chips
Menu2Item3Price=$6.00
#
example1id=
#
#
```

Sample output

```
<Result>
  <Restaurant>Ernies</Restaurant>
  <Menu>
    <Name>Breakfast</Name>
    <Item>
      <Food>Coffee</Food>
      <Price>$2.00</Price>
    </Item>
    <Item>
      <Food>Donuts</Food>
      <Price>$3.00</Price>
    </Item>
  </Menu>
  <Menu>
    <Name>Lunch</Name>
    <Item>
      <Food>Tea</Food>
      <Price>$2.00</Price>
    </Item>
    <Item>
      <Food>Soup</Food>
      <Price>$4.00</Price>
    </Item>
    <Item>
      <Food>Fish and chips</Food>
      <Price>$6.00</Price>
    </Item>
  </Menu>
</Result>

<Result>
  <example>
    <id></id>
  </example>
</Result>
```

Problem I 30 points

This problem has been withdrawn for technical reasons

Problem K Reversing Sequences 100 points

Michael likes to idly play with a row of blocks that he keeps on his desk by picking up a group of them from one end, reversing it, and replacing it at the same end. His objective is to start with a sequence of blocks and use these operations to put them in increasing order from left to right in the smallest possible number of moves. For example, for the blocks 0 2 1 3, this can be accomplished in 3 moves: 02**13** -> 0**231** -> 01**32** -> 0123, where items in **bold** are the ones that are being reversed in that move.

Input will consist of a series of block sequences. Each block sequence will start with the number of blocks (*b*) in the sequence ($1 \le b \le 10$), followed by a colon (:) and a permutation of the integers 0 to *b*-1 representing the initial sequence of the blocks. End of input is indicated by an empty block sequence, i.e. a line specifying b = 0.

For each block sequence in the input, output the initial sequence and the number of moves necessary to order it, as shown in the examples below. All numbers should be separated by single spaces. Note the use of the singular 'move' if there is only one move necessary and the plural 'moves' in all other cases.

Sample input

```
4: 0 2 1 3
3: 0 1 2
3: 1 0 2
0
```

Sample output

```
0 2 1 3 requires 3 moves to put it in order.
0 1 2 requires 0 moves to put it in order.
1 0 2 requires 1 move to put it in order.
```

Problem L Digital Insanity 100 points

One method of identifying a sequence of digits is to apply operations to each element of the sequence, those operations being determined by some given key sequence of operations. In this problem you will be given a sequence of operations corresponding to each digit, and asked to compute the value of the resulting expression when each digit of a number except the last is replaced by that digit and its operation. For example, the operation sequence `++++++++++` would simply result in converting a number to the sum of its digits. If the operation sequence is `+-+--*****` then the number 3456719 becomes 3-4-5*6*7*1-9 = -253.

Notice that the normal precedence rules do not apply — evaluation proceeds left to right. You can assume that all results and all intermediate values will fit into a standard 32-bit signed integer.

So far this problem appears to be the same as problem B (Digital Madness), however, for this problem you are given a series of numbers and the result of the computations carried out on them and it is your task to determine as much of the operation sequence as you can. For example, if you are told that 13 becomes 4, then the only possible operation for 1 is +. If you are told that 22 becomes 4, then 2 cannot be '–', but, since it could be either '+' or '*', in the absence of further evidence it must remain unspecified and be shown as such.

Input will consist of several problem instances. Each problem instance will consist of series of pairs of numbers terminated by a pair of zeroes (0 0). The sequence of problem instances will also be terminated by a pair of zeroes (0 0). The first number of each pair of numbers could contain up to 20 digits, the second will be a signed integer.

Output will consist of the operation sequence, with ?'s replacing operations whose values cannot be determined. You can assume that there will be no inconsistencies, i.e. you will not get something like 13 goes to 4 and 14 goes to –3 in the same set.

Sample input

```
13 4
102 3
22 4
0 0
123 9
245 3
0 0
0 0
```

Sample output

```
Operation sequence 1: ++????????
Operation sequence 2: ?+*?-?????
```

Problems M and N — The Language

The following simple programming language is used in problems M and N.

There are 26 builtin variables/registers that hold 8-bit unsigned integers. The syntax of the language is given below, where '|' denotes choice, '…' means a succession and '(something)+' denotes one or more copies of 'something'.

```
<var> ::= 'a'| ... |'z'                             // names of the variables
<const> ::= '0'| '1'| … |'255'                      // integer constants
<ws> ::= (<space> | <tab> | <newline>)+             // white space
<moper> ::= '+' | '-' | '*' | '/'                   // integer arithmetic operators
<relop> ::= '==' | '!=' | '<' | '>'                 // relational operators
<term> ::= <var> | <const>
<mexpr> ::= <term> <ws> <moper> <ws> <term> | <term>
<bexpr> ::= <var> <ws> <relop> <ws> <const>
<test> ::= <ws> '(' <ws> <bexpr> <ws> ')' <ws>

<assign> ::= 'assign' <ws> <var> <ws> '=' <ws> <mexpr>
<read> ::= 'read' <ws> <var>
<write> ::= 'write' <ws> <var>
<halt> ::= 'halt'

<while> ::= 'while' <test> '{' <statements> '}'
<if> ::= 'if' <test> '{' <statements> '}'
<statements> ::= (<ws> <statement> <ws> ';')+ <ws>

<statement> ::= <assign> | <read> | <write> | <halt> | <while> | <if>
<program> := <statements> 'halt.'
```

You can assume that all programs will be syntactically correct, that all arithmetic operations are safe (for instance, no division by zero), that all variables will be initialised before use and that all arithmetic 'wraps around', i.e. is performed modulo 256.

Problem M Path Test Problem 100 points

Your task as a software engineer is to find a good set of input to test programs. To save on the verification time we want to find the smallest set of input data that will test each reachable execution branch of a program. A simple upper bound on the number of test cases is the number of possible execution paths. Here an execution path is a unique sequence of program statements being executed.

For this problem we want to restrict our attention to loop-free programs (i.e., those programs without the <while> statement). You may assume that each Boolean expression (<bexpr>) may evaluate to either 'true' or 'false' for some input. (That is, we can assume that each branch of an <if> statement may be taken irrespective of any logic.)

Input will consist of a series of loop-free programs. Each program will be syntactically correct. Successive programs are separated by a line containing only a single #. The sequence of programs will be terminated by an empty program, i.e. by another line containing only a single #.

Output will consist of a single line for each program in the input, containing the total number of potential different 'reachable paths' of execution.

Sample input

```
read a ; read b ;
if ( a < 5 ) { if ( b < 5 ) { write a ; } ; write b ; } ;
halt.
#
read a ; assign b = a ;
if ( a < 5 ) { if ( b < 5 ) { write a ; } ; read b ; } ;
assign c = a ; if ( c < 2 ) { write c ; } ;
halt.
#
#
```

Sample output

```
3
6
```

Problem N The Halting Problem 100 points

The "Halting Problem" is the problem of deciding (in a reasonable time) whether a program will halt or run forever. This problem is known to be unsolvable in general. However for some programs it is relatively easy to tell from the text whether it will halt. For the programming language specified above it is possible to decide whether a given program, plus input, will halt.

For this problem we will restrict our attention to programs that use only the first three 8-bit unsigned integer variables 'a', 'b' and 'c'.

The input will be a sequence of programs and data. The data part is a sequence of integers on a single line that precedes each program listing. Successive programs are separated by a line containing only a single #. The sequence of programs will be terminated by an empty program, i.e. by another line containing only a single #.

For each program we want to know if the program 'halts', 'loops' or runs out of input data ('runtime error'). For each program output one of these strings, as appropriate, on a separate line.

Sample input

```
3 4 2 -1
read a ; read b ;
if ( a < 5 ) { if ( b < 5 ) { write a ; } ; write b ; } ;
halt.
#
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 -1
read a ;
while ( a < 5 ) { write a ; read b ; } ;
halt.
#
9 100 -1
read a ; read b ;
while ( a < 10 ) { while ( b > 1 )
     { assign b = b / a ; } ; } ;
halt.
#
#
```

Sample output

```
halts
runtime error
loops
```