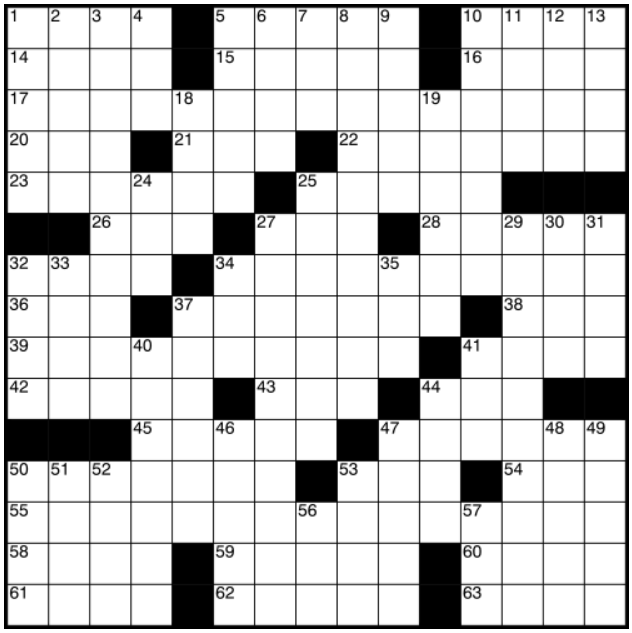
University of Manitoba
Open Programming Contest
September 22, 2012

General Instructions:

1. Submit solutions using the PC^2 software.
2. The questions are **not** listed in order of difficulty. Some questions will be easier than others, and these are not necessarily the problems listed first.
3. All input should be read from standard input

Problem 1 – Crossword Numbering

Crossword puzzles are grids of white and black spaces where the contents are filled with letters so that horizontal and vertical sections correspond to words indicated by clues. The clues are numbered and written separately from the puzzle. All of the puzzles will be square grids. Here is an example of a puzzle:



Notice the small numbers in the corner of the cells only appear when a word in either the horizontal (left-to-right) or vertical (top-to-bottom) should start in that cell. For instance the number 1 indicates that a word of length four should start in the upper-left-hand corner of the puzzle and move to the right. The number 4 (in the cell six spots from upper-left-hand corner in the top row) indicates that a word should start in both the horizontal and vertical directions.

The numbers start in the upper-left hand corner and proceed in row-major order (i.e., along the first row, then subsequent rows), always from left to right. If no clue starts at a position, then there is no number.

Given a blank crossword puzzle consisting of only blanks (indicated by an underscore _) and black squares (indicated by a hash mark #), your program should write in the numbers of the clues. If no clue starts at a position, an underscore should continue to represent the cell. All black squares should remain as hash marks.

Start your numbering with numbers 0-9 (clue numbers should always start with zero, clearly). After 9, write the letters A-Z to indicate clue numbers 10-35, and

a-z to indicate the clue numbers 36-61. Clue number 62 should be indicated by @, the at sign. No puzzles will have more than 63 clues (numbered 0-62).

Input

The first line of the input gives the number of puzzles to consider. Each puzzle is first described by a number n on a line, which describes the size of the puzzle, then n lines, each with n characters per line. All characters are either _ or #.

Output

For each puzzle, write "Case m" where m is the case number, starting with zero. Then print out the numbered puzzle.

Sample Input	Sample Output
2 5 __#__ __#__ #___# _____ __#__ 15 _____#____#____ _____#____#____ __________#____ ##_____##______ ___#__________ ______#_____### _____#_____#___ ____#_____#____ ___#_____#_____ ###_____#______ __________#___ ______##_____## ____#_________ ____#_____#____ ____#_____#____	Case 0: 01#23 4_#5_ #67_# 8___9 A_#B_ Case 1: 01234#5678#9ABC D____#E___#F___ G____H____#I___ ##J____##KL____ MN_#O__PQ______ R__S__#T____### U____#V____#WXY Z___#a____#b___ c__#d____#e____ ###f____#g_____ hij_____k__#l__ m_____##n__o_## p___#qrs_____tu v___#w____#x___ y___#z____#@___

Problem 2 – A-S-P-E-C-T. Find out what it means to me.

Alice is buying a new TV. The main measurement she is concerned with is the total area of the TV. But when buying a TV, only the aspect ratio and the diagonal are given. The diagonal is the measurement from one corner of the TV to the opposite corner. The aspect ratio is of the form x:y, where if the width of the TV is x units, then the height is y units (for some appropriate unit, which is unknown).

Given the aspect ratio and the diagonal length (in cm), output the area of the TV screen.

Input

The first line of input is a single number N, which gives the number of TVs. The next N lines each give first the aspect ratio as X:Y, where X and Y are at most 32-bit integers, followed by a single space, then the length of the diagonal Z (in cm). Z is also a 32-bit integer.

Output

Output the area of the TV in cm^2 for each TV, on its own line. Round the result to the nearest cm^2.

Sample Input	Sample Output
2 16:9 100 15:8 34	4272 480

Problem 3 – Balancing Act

A balanced parenthesis expression is a sequence of opening and closing parentheses, given by the characters “(“ and “)”, where each opening parenthesis is matched with a closing parenthesis. For instance, the sequences `()()` and `(())()` are balanced, but `)(`, `(()` and `()))` are not.

A partially specified parenthesis expression is a sequence of the characters "`(`", "`)`" and "`.`" (the period character). There may be several ways to replace a period with `(` or `)` in a partially specified parenthesis expression to form a balanced parenthesis expression: for instance, the expression `..)...` can be completed as

`(())()` or `(())()`.

The expression `...(....` can be completed in seven ways:

`((()))` `()(())` `()((()))` `()(()())` `(()()())` `(()(()))`
`(()())()`

Given a partially specified parenthesis expression, give the number of ways to complete it.

Input

The input consists of partially specified parenthesis expressions, one per line. Each line has at most seventy characters.

Output

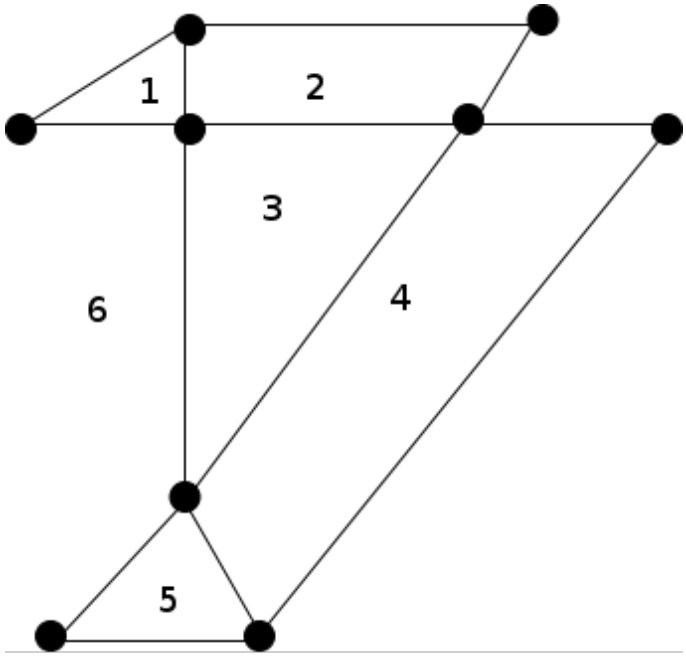
For each partially specified parenthesis expression, output the number of ways it can be completed into a balanced parenthesis expression

Sample Input	Sample Output

`..)...`
`...(....`
`...`
`...(`
	`2`
	`7`
	`0`
	`0`

Problem 4 – FaceTime

A plane graph is a set of points and edges between pairs of points that has no lines crossed. When you draw a plane graph, it defines a set of faces: a face is an area that is bounded by a set of edges. We count the exterior face as a face too: it is the face that consists of the infinite region outside the graph. For instance, the following graph has nine points (indicated by solid dots), twelve edges (indicated by the lines between points) and six faces, indicated by the numbers in the picture:



Given a graph defined by a set of points in two-dimensional space, as well as a set of edges between the points, how many faces are there?

Input

Each graph starts with a line "GRAPH X" where X is a number. Following this is a list of points, each given as a label, followed by an x and a y coordinate, all on a single line. The labels for the points will be pN where N is a number starting from 1. All points will be unique. There will always be at least two points in each graph.

Following the points is a list of the edges, with each edge on its own line. The edges are given by a label, followed by the pair of points that the edge connects. Edges are not oriented, and at most one edge connects any pair of points. Labels of the edges will be of the form eN where N is a number starting from 1.

All of the graphs are connected, meaning there is a path from any vertex to any other vertex.

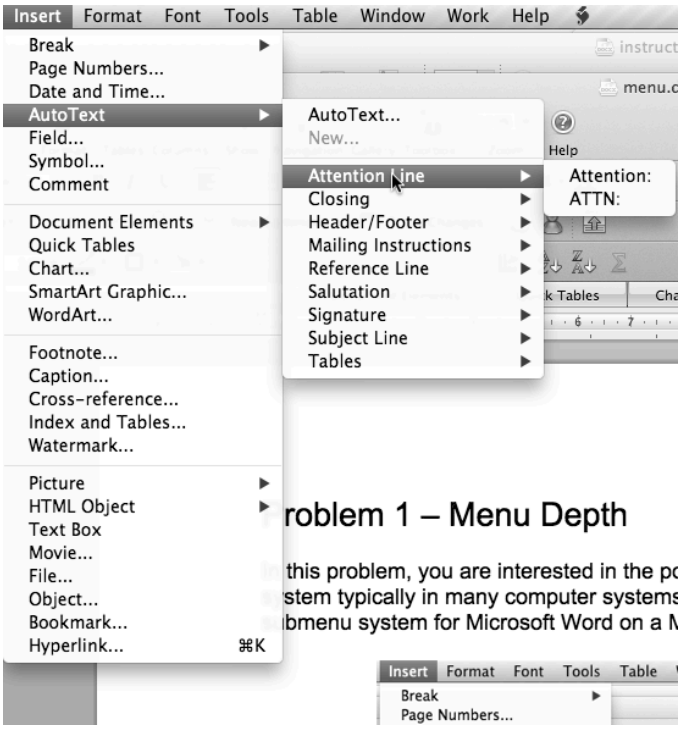
Output

For each graph, output "GRAPH X: y faces." where y is the number of faces. You can always output "faces", even if the answer is 1 face.

Sample Input	Sample Output
GRAPH 1 p1 1 1 p2 2 1 p3 3 2 e1 p1 p2 e2 p2 p3 e3 p1 p3 GRAPH 2 p1 1 1 p2 3 1 p3 2 2 p4 1 3 p5 2 3 p6 4 3 p7 6 3 p8 2 4 p9 5 4 e1 p4 p8 e2 p5 p8 e3 p4 p5 e4 p8 p9 e5 p5 p6 e6 p6 p9 e7 p3 p6 e8 p6 p7 e9 p2 p7 e10 p1 p3 e11 p3 p2 e12 p1 p2 e13 p3 p5	GRAPH 1: 2 faces. GRAPH 2: 6 faces.

Problem 5 – Menu Depth

In this problem, you are interested in the position of a menu in a menu-submenu system typically in many computer systems. Here is a screen shot from a menu-submenu system for Microsoft Word on a Mac:



Notice that the Autotext menu item has a submenu (indicated by the right-facing triangle on the left-side of the menu box. In Autotext's submenu, "Attention Line" has a submenu, which itself has two menu items.

We specify a menu as a plain-text file. Each item is preceded by some number (possibly zero) of dashes. After each item X is its submenu, if it exists, including subsubmenus, etc., before the next item in the same menu as X. If an item is preceded k dashes, all the items in its submenu are preceded by k+1 dashes. The top-level menu items are preceded by zero dashes.

For instance, the plain-text for the menu depicted above would begin with

```
Break
-Page Break
-Column Break
PageNumbers...
Date and Time...
AutoText
-AutoText...
-New...
-Attention Line
--Attention:
--Attn:
```

(Note that the submenu for "Break" is not displayed in the picture. Assume it has only items called "Page Break" and "Column Break" without any subsubmenus.)

In this question, you want to calculate the distance between the beginning of a menu (in this case, the top of the word "Break") and the menu item that is the furthest away from that point. We use the following conventions:

- each character is 1 unit high and 1 unit wide, regardless of what character it is.
- the width of a menu is exactly the width of its longest menu item. (for instance, in the specification above, the width of the submenu of "Break" is 12 units, as this is the length of the string "Column Break"
- a submenu is placed with its first element at the same vertical distance from the top of the menu as its parent element (for example, "Page Break" is on the same horizontal line as "Break", its parent, in the image).
- the position of a menu item is given by its upper-left hand corner. So in the example above, "Break" is at position (0,0), while "Page Break" is at position (W,0), where W is the width of the menu containing "Break", and "Page Numbers…" would be at position (0,1).
- the distance between a point (p,q) and (0,0) is measured in the usual way, i.e., $\sqrt{p^2+q^2}$.

Input

Each menu begins with an identifier line of the form "#Menu n" where n is an integer beginning with 1. Each subsequent line gives on menu item, until the next identifier line is encountered. Menu-submenu structures are as specified above. All menu items will consist of the characters A-Z,a-z, digits 1-9, spaces and punctuation (.,:;) only. Menu items will not contain dashes (-) or hash marks (#). All menu items will have one or more characters. The first line of a menu will be at the root level (i.e., it will have no dashes before it). Labels of menu items are not necessarily unique.

Output

For each menu, write the output: "#Menu n: X units" where n is the case number and X is the number of units, rounded to the nearest tenth of an integer.

Sample Input	Sample Output
#Menu 1 A B C #Menu 2 A -BB --C #Menu 3 A BB -C	#Menu 1: 2.0 units. #Menu 2: 3.0 units. #Menu 3: 2.2 units

Problem 6 – Negalomania

The binary system is a way of denoting numbers in base-2, by using the digits 0 and 1. So if the binary representation of a (positive) number n is $b_1\ b_2\ b_3\ \ldots\ b_k$, which represents that

$$n = b_1\ 2^{k-1} + b_2\ 2^{k-2} + \ldots + b_{k-1}\ 2^1 + b_k\ 2^0.$$

For example, 15 in binary is 1111 since

$$15 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 8 + 4 + 2 + 1.$$

We can use other bases as well, like base-3 (ternary), base-10 (decimal, the usual representation of numbers) or base-16 (hexadecimal). In each case, we replace powers like 2^i in the sum above with b^i, where b is the base. We also use digits 0,1,...,b-1.

But there's nothing that says we need to use a positive number as the base. In this question, we ask you to consider "negabinary" (base-(-2)) representation of numbers. In this representation, the base is -2 but the digits that can be used are still {0,1}. So the representation of the positive number n will be $b_1\ b_2\ \ldots\ b_k$ where

$$n = b_1\ (-2)^{k-1} + b_2\ (-2)^{k-2} + \ldots + b_{k-1}(-2)\text{^}1 + b_k\ (-2)\text{^}0.$$

The only thing that makes this not completely insane is the fact that if p is an even integer, $(-2)^p$ is a positive number. For example, 15 in negabinary is 10011, since

$$1\ (-2)^4 + 0\ (-2)^3 + 0\ (-2)^2 + 1\ (-2)^1 + 1\ (-2)^0$$
$$= 16 + 0 + 0 + (-2) + 1 = 15.$$

The process for converting a number from decimal to negabinary is the same as converting from decimal to binary, except that we are computing with respect to the base -2 rather than +2. Note that (when programming) x mod -2 will be negative if x is a negative odd number, but if x = -2 c - 1, then x = (-2) (c+1) + 1 as well.

Input

The input file consists of several lines of input, with one number per line. Each number is greater than zero and strictly less than 2^{63}. The last line of the file is zero; this number should not be processed.

Output

For each input value, output its negabinary equivalent on a line.

Sample Input	Sample Output
1 2 15 0	1 110 10011