Important Contest Instructions!!

Please *read the following instructions carefully. They contain important information on how to run your programs and submit your solutions to the judges. If you have any questions regarding these instructions, please ask a volunteer before the start of the competition.*

Program Input
Most programs will require input. You have two options:

1) Your program may read the input from a file. The input data will be in the local directory in the file **probXX.txt**, where 'XX' is the problem number.

2) Your program may read the input from the keyboard (standard in). You may type everything on the keyboard, or you may copy the data from **probXX.txt** into the standard in. ***Tip:*** *Type 'Ctrl-Z <return>' to signal the end of keyboard input.*

   Note: An easy way to enter keyboard data is by redirecting the contents of a file to your program. For example, if you are executing prob01, the input file **prob01.txt** can be redirected to the standard in of your program using syntax like this (examples are shown for each of the allowed languages):

```
%> java prob01 < prob01.txt
%> java –jar js.jar prob01.js < prob01.txt
%> python prob01.py3 < prob01.txt
%> prob01.exe < prob01.txt
```

   Your program will behave exactly as if you were typing the input at the keyboard.

Program Output
All programs must send their output to the screen (standard out, the default for any print statement).

Submitting your Programs

Interpreted Programs (Java, JavaScript, Python.) Your program must be named probXX.java / probXX.js / probXX.py2 / probXX.py3, where 'XX' corresponds to the problem number. For Python, use the extension that matches the Python version you are using. Please submit only the source (.java, .js, .py2 or .py3). For java, the main class must be named probXX. Note there is no capitalization. All main and supporting classes should be in the default (or anonymous) package.

Native Programs (C, C++, etc.) Your program should be named probXX.exe, where 'XX' corresponds to the problem number.

You are <u>strongly</u> encouraged to submit solutions for Problems #0 and #1 (see next pages) <u>prior</u> to the start of the competition to ensure that your build environment is compatible with the judges' and that you understand the Input and Output methods required.

Problem 0 Welcome! [1 point]

NOTE – this is the 1st of two problems that can be solved and submitted before the start of the CodeWars competition. Teams are **strongly** encouraged to submit this problem **prior** to the start of the competition – hey, it's basically a free point!

Summary

The sole purpose of this problem is to allow each team to submit a test program to ensure the programs generated by their computer can be judged by our judging system. Your task for this program is a variation on the classic "Hello World!" program by saying hello to our two newest CodeWars sites, Barcelona and Newcastle. All you have to do is print "Welcome, Barcelona and Newcastle!" to the screen.

Output

```
Welcome, Barcelona and Newcastle!
```

Problem 1 Greetings [1 point]

NOTE – this is the 2nd of two problems that can be solved and submitted before the start of the CodeWars competition. Teams are **strongly** encouraged to submit this problem **prior** to the start of the competition – hey, it's basically a free point!

Summary

You'll have no chance to win at HP CodeWars (or life) if you don't know how to do Input and Output properly. You also won't do well at CodeWars if you are rude to your judges.

Write a program to greet your esteemed judges appropriately. Read in the name of a judge and output your greeting in the appropriate format.

If you're confused at this point, go back and re-read your contest instructions.

Input

The input will be your judge's first name, a single word with no spaces:

```
Simon
```

Output

Welcome your judge with a friendly, creative greeting of some sort that includes the judge's name (does not have to match the below example):

```
Greetings, Simon the Great!  I genuflect in your general direction!
```

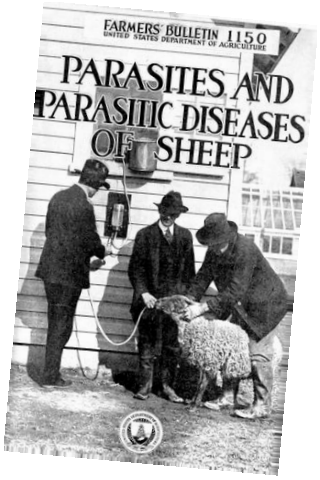
Problem 2 Parasitoid / Host Interaction Models [2 points]

Summary

W. R. Thompson (1924) was interested in what happened to parasitoid/host populations in areas where parasites were released into an area with a large host density. In the model initially he assumed that the parasitoid would only lay one egg in each host that was found. Thus:

```
(No. eggs laid) = (Mean Female Egg Compliment)(No. Females Searching)
            Pe = C X P
```

However, this model did not work too well because many parasites are unable to distinguish between parasitized and unparasitized hosts. Thus Thompson's model predicted a higher rate of parasitism than would actually occur. In reality, a host can end up with more than one parasitoid egg and is then "superparasitized".



Thompson's model may not have been very accurate, but it makes a great CodeWars program. Write a program that uses Thompson's model to predict the rate of parasitism for a pair of input values.

Input

The first line of input is the value of C, the Mean Female Egg Compliment, an integer between 1 and 10,000. The second line is the value of P, the Number of Females Searching, an integer between 1 and 100,000.

```
1300
97450
```

Output

The program must print the value of Pe, the Number of Eggs Laid. The answer must match the judge's expected value precisely.

```
126685000
```

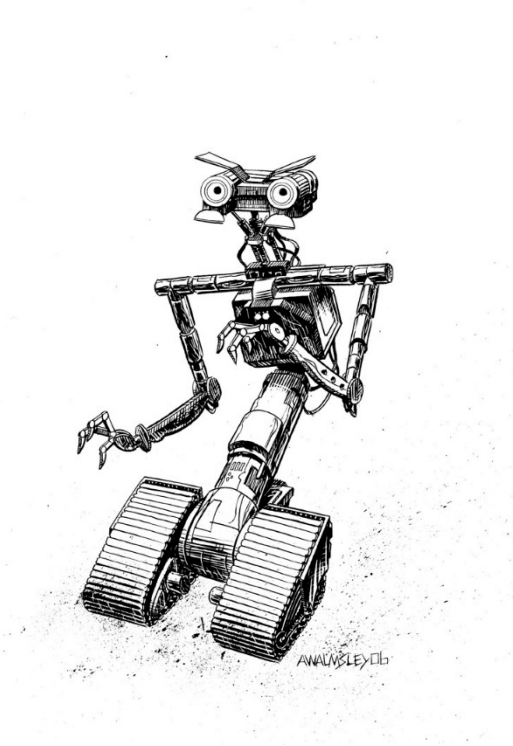
Problem 3 No Disassemble! [3 points]

Summary

There's a short circuit in the management fabric induced by a power surge from a lightning strike. The engineering team needs you to take an unmarked white van and investigate the numbered input nodes in the fabric.

You must inspect each node by disassembling it. If it is working, you'll send a message back to the laboratory at corporate headquarters and reassemble the node. With the fabric not working correctly you'll be using a low-frequency channel that can only send very short messages. In fact, you can only send a single number.

The management at corporate headquarters would like to see a human-friendly message instead of a single number. So before leaving, you must write a program that reads a single number and writes a friendly message on the screen.



Input

The input is a single integer between one and ten, inclusive.

Example 1:
```
5
```

Example 2:
```
10
```

Output

The program must print a sentence in the format shown in the examples below, using the input number written as a word.

Example 1:
```
Number five is alive!
```

Example 2:
```
Number ten is alive!
```

Problem 4 NetOS App [3 points]

Summary

The IQ Computer Corporation* has recently acquired Hand Computing* and plans to release a new line of products based on Hand's innovative technology. IQ has rebranded HandOS with the name netOS and is preparing for product launch of its newest devices.

There's just one catch: netOS doesn't have nearly as many apps as those other guys.

Never ones to worry, the IQ marketing team has decided to offer free netOS devices to the first 1,000 software developers who submit apps for the new platform. Being a savvy software developer, you have decided to write a simple app that calculates the area of a room. The user just enters two integers for the length and width of a room and the app will display the area.

If your conscience starts to bother you because you're writing such a simplistic app, don't fret. As it turns out, IQ Computer Corporation has no intention of actually publishing any of the apps submitted, so writing a useful app would just be a waste of everyone's time. This is how multinational corporations roll.

Besides, it's not like 1,000 more apps would make a difference anyway.

Input

Each line of the input is two integer values. The last line of input is two zeros.

```
15 9
11 8
0 0
```

Output

For each line of input, the program must multiply the two input numbers and print the result.

```
135
88
```

* NOTE -- The corporations appearing in this work are fictitious. Any resemblance to real corporations, profitable or bankrupt, is purely coincidental.

Problem 5 Distributive Law [3 points]

Summary

In mathematics, the Distributive Law says

$$A \times (B + C) = A \times B + A \times C$$

Write a program to demonstrate the Distributive Law.

Input

There will be three lines of input. The first line has the value for A, the second for B, and the third for C. All values will be positive integers.

```
11
5
4
```

Output

The program must print three lines demonstrating the Distributive Law with the input values. The first line must substitute the values into the equation. The second line must show the result of the first level of evaluation. The third line must show the result of the final evaluation. Follow the pattern shown below.

```
11 x (5 + 4) = 11 x 5 + 11 x 4
11 x 9 = 55 + 44
99 = 99
```

Problem 6 Dirt Yard [3 points]

Summary

In the United States, landscaping companies sell various types of soil and gravel by the "yard". This is an abbreviation for a cubic yard, i.e., a cube that is one yard long, one yard wide, and one yard deep.

Now this bit of trivia is important to a suburban gardener who wishes to grow a grape vine in their (ahem) back yard. Grape vines require soil with good drainage, so if the gardener's soil does not drain well they might decide to dig a hole and fill it with a mix of sand and topsoil. Plant spacing is typically published in feet, so the gardener is likely to plan the size of the hole in feet, but needs to order the dirt in “yards.”

Write a program to compute the number of (cubic) yards of soil required to fill a hole that is measured in feet. There are three feet in a yard, which equates to twenty-seven cubic feet per cubic yard.

> *To forget how to dig the earth and to tend the soil is to forget ourselves.*
>
> *– Mahatma Gandhi*

Input

The input consists of three lines, each with a single integer. These are the length, width, and depth of the hole, measured in feet.

```
4
6
3
```

Output

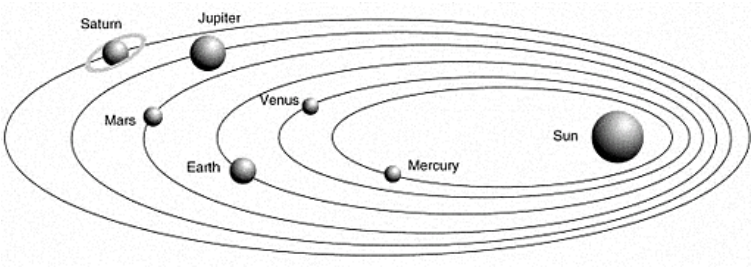
The program must print the minimum number of (cubic) yards of soil required to fill the hole. Landscaping companies sell soil in whole yards, not in fractional parts, so the answer must be the smallest integer number of cubic yards that will fill the hole.

```
3
```

Problem 7 Kepler's Third Law [4 points]

Summary

Johannes Kepler derived three mathematical laws for planetary motion in the early 1600's. This work was based on measurements made before the advent of the telescope using instruments called quadrants and sextants. Oh, and before the advent of the computer as well.



Kepler's third law of Planetary Motion captures the relationship between the distance of planets from the Sun and their orbital periods. It states that the square of the orbital period of a planet is proportional to the cube of the semi-major axis of its orbit. If we assume (for convenience) that the period P is measured in Earth-years and the orbital radius R is measured in Astronomical Units (AU), then the equation is simply this:

Mathematically:

$$P^2 = R^3$$

where P is the period measured in Earth-years, and R is the semi-major axis (orbital radius) in Astronomical Units (AU).

Write a program to calculate the orbital radius of a planet given its orbital period. To solve this problem you may need to know that you can express square roots and cube roots as *fractional exponents*. For example:

$$x^{1/2} = \sqrt[2]{x}$$

$$x^{1/3} = \sqrt[3]{x}$$

Input

Each line of input is the orbital period (P), in years, of a body orbiting the sun. The input ends with a value of zero.

```
1.8808
4.60
0.615198
30.07
0
```

Output

For each period, the program must print the semi-major axis (R) in AU. The answer must be accurate to within +/- 0.01 AU.

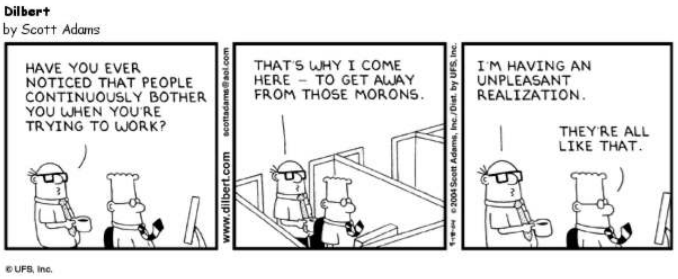
```
1.523679
2.7668
0.723327
9.6699
```

Problem 8 Counting Cubes [5 points]

Summary

We've just opened up a new facility that has 300 new spaces available for our employees. We're now in the process of identifying the employees that are moving into this new space. We have a cube assignment list and want to ensure that we're using the space effectively. In order to know that, we need to find out if there are any empty cubes, any duplicate cube assignments, and any unassigned cubes. Write a program that can analyze the current list and provide answers to those questions.

The first line of input will contain the number of employee to cube mappings that follow. Each employee to cube mapping will consist of the employee's first name followed by a single space and a cube location (0 to 300). An employee with cube location of 0 does not have an assigned cube. If a cube is empty the employee name will be "NA". The employee names are unique and we have ensured that each employee is listed only once.



The output will consist of the number of empty cubes, the number of duplicate cube assignments (*two or more* employees assigned to the same cube), and the number of unassigned cubes.

Sample Input

```
16
Joe 11
Bob 123
NA 101
Katy 125
Sam 47
Mike 59
NA 23
Vivek 62
Fred 0
Lars 74
Oscar 86
Caroline 11
NA 90
Erin 11
Rachel 111
Nate 125
```

Sample Output

```
Empty Cubes: 3
Duplicate Cube Assignments: 2
Employees without Cube: 1
```

Problem 9 A Sorted Affair [6 points]

Summary

Q: How do you win HP CodeWars?
A: Everyone's a winner at HP CodeWars.

Seriously, if you want to win you have to score the most points, right? How do you think the judges keep track of the points? One simple way would be to record the team number and point value every time a program is judged to be correct. Then at the end of the contest a program could calculate each team's total and print the top five teams. Doesn't that sound easy?

Write a program to determine the top five teams using raw score data.

Input

Each line of input represents a single correct program using two integer values: a three-digit team number and a point value between one and twenty-one. The end of input is signaled by two zeroes.

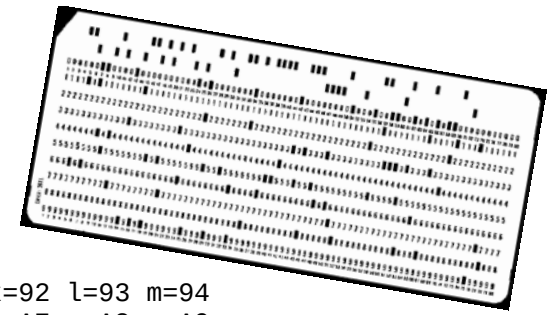
```
123 1                (continued...)
354 2                481 1               354 4
213 1                987 3               742 2
354 5                246 1               508 3
112 2                213 2               742 3
508 1                354 6               642 3
481 5                742 1               213 5
508 5                354 13              508 2
481 3                987 15              354 3
354 7                213 8               642 1
508 9                987 7               213 11
112 3                833 1               0 0
```

Output

The program must print the top five teams, in order. Each line of output must have the team rank, the team number, and the total number of points scored by that team. Lucky for you there will be no ties.

```
1 354 40
2 213 27
3 987 25
4 508 20
5 481 9
```

Problem 10 EBCDIC [7 points]



Summary

Extended Binary Coded Decimal Interchange Code (EBCDIC) is an 8-bit character encoding used mainly on IBM operating systems. EBCDIC was devised in 1963 and 1964 by IBM and was descended from the 6-bit code used with punched cards of the late 1950's and early 1960's.

EBCDIC Hexadecimal values

```
a=81 b=82 c=83 d=84 e=85 f=86 g=87 h=88 i=89 j=91 k=92 l=93 m=94
n=95 o=96 p=97 q=98 r=99 s=A2 t=A3 u=A4 v=A5 w=A6 x=A7 y=A8 z=A9
A=C1 B=C2 C=C3 D=C4 E=C5 F=C6 G=C7 H=C8 I=C9 J=D1 K=D2 L=D3 M=D4
N=D5 O=D6 P=D7 Q=D8 R=D9 S=E2 T=E3 U=E4 V=E5 W=E6 X=E7 Y=E8 Z=E9
space=40  period=4B  comma=6B exclamation=5A
```

A different character-encoding scheme was also first published in 1963, called the American Standard Code for Information Interchange (ASCII). ASCII is a 7-bit code that was adopted for a wide range of computer systems and became the most commonly used character encoding on the internet. It was surpassed in 2007 by UTF-8, which uses ASCII as a subset. Your laptop, desktop, tablet, and phone almost certainly use ASCII encoding.

> DUDE: I thought they were just letters and stuff.
> TECHIE: Dude, each letter is represented by a number. It's called a code.
> DUDE: (nodding slowly) oh.....

But sometimes old technology lingers and to this day there are systems that store and retrieve data using EBCDIC. Modern mainframes (such as IBM zSeries) include processor instructions, at the hardware level, to accelerate translation between character sets.

Write a program to convert text from EBCDIC to ASCII.

Input

The first line of input will indicate the number of lines of EBCDIC. Each line begins with a decimal number of EBCDIC codes, followed by that number of EBCDIC codes.

```
3
12 C8 D7 40 C3 96 84 85 E6 81 99 A2 5A
17 E5 85 95 89 6B 40 A5 89 84 89 6B 40 A5 89 83 89 4B
21 E6 85 40 81 99 85 40 A3 88 85 40 83 88 81 94 97 89 96 95 A2 4B
```

Output

The program must print the text represented by the EBCDIC input. Each line of input should correspond to a line of output.

```
HP CodeWars!
Veni, vidi, vici.
We are the champions.
```

Problem 11 Diamond in the Rough [7 points]

Summary

A "diamond in the rough" is someone or something having desirable qualities or potential but lacking experience, refinement or polish. When interviewing college graduates, technology companies usually make an effort to distinguish the diamonds in the rough from the lumps of coal.

Write a program that can generate diamond patterns of different sizes.

Input

Each line of input contains three positive integers: the size of the diamonds and the number of rows and columns of diamonds to be drawn. Due to the way the diamonds are drawn, the diamond size will always be an even integer. All values will be less than one hundred. The input ends with three zeros.

```
6 2 4
2 3 7
0 0 0
```

Output

The program must print diamonds in a rectangular grid using the slash and backslash characters for the bodies of the diamonds. A diamond of size two has exactly two slashes and two blackslashes arranged in a diamond pattern. For diamonds larger than size two, the program should fill the spaces between the diamonds with hash characters to represent lumps of coal. The diamond grid must match the input rows and columns.

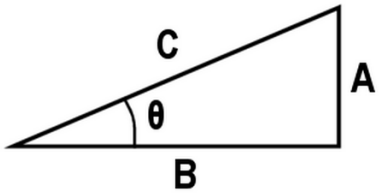
```
##/\####/\####/\####/\##
#//\\##//\\##//\\##//\\#
///\\\///\\\///\\\///\\\
\\\///\\\///\\\///\\\///
#\\//##\\//##\\//##\\//#
##\/####\/####\/####\/##
##/\####/\####/\####/\##
#//\\##//\\##//\\##//\\#
///\\\///\\\///\\\///\\\
\\\///\\\///\\\///\\\///
#\\//##\\//##\\//##\\//#
##\/####\/####\/####\/##


/\/\/\/\/\/\/
\/\/\/\/\/\/\
/\/\/\/\/\/\/
\/\/\/\/\/\/\
/\/\/\/\/\/\/
\/\/\/\/\/\/\
```

Problem 12 Angle of Repose [8 points]

Summary

In Geology, the angle of repose is the steepest angle at which dry, unconsolidated sediment is stable. Different materials, like sand and snow, have different angles of repose. A material's angle of repose can be compared with the actual slope angle of a pile as part of an analysis of the risk of avalanche or landslide.

Write a program to calculate the maximum slope angle of a pile of material. To do this, you'll need to apply some basic trigonometry. Remember that you'll need to use the arcsin function to calculate the angle theta and that arcsin may return the angle in radians, not degrees. Please consult your language documentation. You did bring documentation, right?



$$A^2 + B^2 = C^2$$

$$\sin\theta = \frac{A}{C}$$

Input

The input is a two-dimensional square grid that represents the height of a pile of material measured at regular one-meter intervals (as seen from above). The value at each point in the grid is the height of the material, in meters, at that location. The first line of the input is an integer that indicates the number of row and columns in the grid. You should not assume a certain number of spaces between the numbers or assume that the decimals will align.

```
9
  4.5  4.6  4.4  4.5  4.5  4.4  4.5  4.6  4.5
  4.5  4.6  4.6  4.7  4.8  4.6  4.5  4.5  4.4
  4.4  4.5  4.7  4.9  5.0  4.8  4.7  4.6  4.5
  4.5  4.7  4.8  5.0  5.2  5.1  5.0  4.8  4.6
  4.5  4.6  4.8  5.0  5.3  5.1  4.9  4.8  4.6
  4.4  4.6  4.9  5.1  5.2  5.1  4.9  4.7  4.5
  4.5  4.5  4.6  4.8  4.9  4.8  4.7  4.5  4.4
  4.6  4.6  4.5  4.6  4.7  4.7  4.6  4.5  4.4
  4.5  4.5  4.4  4.5  4.5  4.6  4.5  4.4  4.5
```

Output

The program must print the steepest slope angle (in degrees) between any adjacent points (including the diagonals) on the grid. The answer must match the expected value to within plus or minus 0.1 degrees.

```
Max angle is 19.47 degrees
```

Problem 13 Non-Repeating Binary [10 points]

Summary

Our competition has created a new lock that opens simply by properly raising and lowering a lever the right number of times, then pressing a button. They've encoded their combination as a series of decimal digits. When the digits are gathered in the right combinations, the BINARY representation of the number is an alternating series of 1 and 0.

For example, the combination 2730341 can be split into two numbers:
2730 (binary 101010101010) and 341 (binary 101010101).

We need a program that can read their combination and split it into numbers that have an alternating-binary representation. Each number will be greater than zero and less than 1 million.

Input

The first line of input is the combination as a string of decimal digits, ending with a period. The second line of input is a single integer, N.

```
Example 1:
21218453415461109221.
6

Example 2:
542.
2
```

Output

Print the N integers in the order of the original string, each on its own line.

```
Example 1:
21
21845
341
5461
10922
1

Example 2:
5
42
```

Problem 14 Squaring Rectangles [11 points]

Summary

Our new company “Competitive Carpets”, or C-squared for short, is all about customer comfort. We have an extra unique flair because we pride ourselves on conscientious covering of any room using only carpet cut in the shape of squares. In fact, we will finish a room with the fewest pieces of carpet needed.

Our carpet squares all measure an integer number of feet on each edge, and we only accept jobs for rooms with both width (W) and length (L) an integer number of feet. Obviously if a room is perfectly square, we can finish the job with a single piece of carpet. And the maximum number of carpet squares we would ever need is LxW squares, each 1-foot on a side. But we can always find a smaller number.

Input

The input is one line with two integers, the Width and Length of the room. For this problem, the maximum for each is 25.

Example 1:
```
11 10
```

Example 2:
```
19 18
```

Output

On the first line, print “N squares can cover WxL” (replacing N, W, and L with their values). On the next line print the sizes of the squares in increasing order, separated by spaces. Finally, print a map of the room, using a different lowercase letter (a, b, …) for each square of carpet used. (No problem will ever need more than 26 squares.) Your map may not match the sample output, but your minimum number of squares must.

Example 1:
```
6 squares can cover 11x10.
2 2 4 5 5 6
aaaaaabbbbb
aaaaaabbbbb
aaaaaabbbbb
aaaaaabbbbb
aaaaaabbbbb
aaaaaaccccc
ddddeeccccc
ddddeeccccc
ddddffccccc
ddddffccccc
```

Example 2:
```
7 squares can cover 19x18.
3 5 5 7 7 8 11
aaaaaaaaaaabbbbbbbb
aaaaaaaaaaabbbbbbbb
aaaaaaaaaaabbbbbbbb
aaaaaaaaaaabbbbbbbb
aaaaaaaaaaabbbbbbbb
aaaaaaaaaaabbbbbbbb
aaaaaaaaaaabbbbbbbb
aaaaaaaaaaabbbbbbbb
aaaaaaaaaaacccddddd
aaaaaaaaaaacccddddd
aaaaaaaaaaacccddddd
eeeeeeeffffffddddd
eeeeeeeffffffddddd
eeeeeeeffffffggggg
eeeeeeeffffffggggg
eeeeeeeffffffggggg
eeeeeeeffffffggggg
eeeeeeeffffffggggg
```

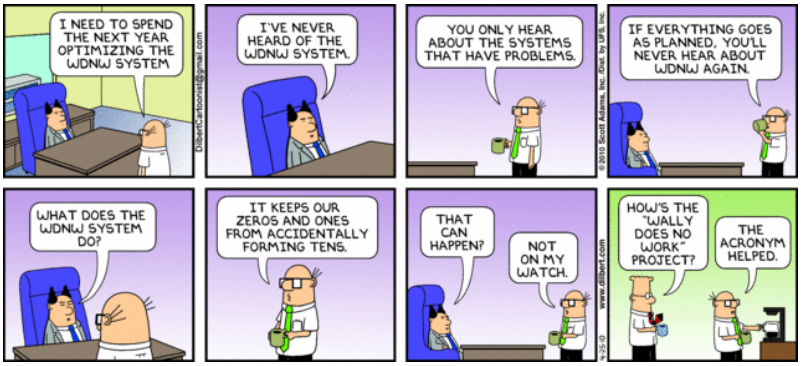
Problem 15 Analysis of Acronyms [11 points]

Summary

An acronym is an abbreviation made from the initial letters of the words in a phrase. For example, the National Aeronautics and Space Administration (NASA) uses acronyms for various projects and processes, like the Automated Retrieval and Tracking System (ARTS), Certificate of Flight Readiness (COFR), and Time Duration of Burn (TDB).

You may have noticed that sometimes (but not always) some initial letters are not used in the acronym (compare COFR with TDB). When all the words are represented, the acronym is complete. If one or more words is not represented, the acronym is partial.

The formation of acronyms can be even more interesting when more than one initial letter of a word is used, as in the Canadian space telescope called Microvariability and Oscillations of STars (MOST). For the purposes of this contest we'll only consider the first two letters of a word, even though humans are far more creative. When two initial letters from a word are used, the acronym is complex. When only initial letters are used the acronym is simple.



Write a program that can analyze a phrase and determine if a given word is an acronym, and if so, what type.

Input

The first line of input provides the number of phrases to be analyzed. Each line after consists of an acronym (candidate) followed by a phrase. The acronyms and phrases may be mixed (upper and lower) case. Words may be hyphenated.

```
7
SPBSG slowly pulsating B super-giants
GNU GNU is not Unix
COSTAR corrective optics space telescope axial replacement
MOST Microvariability and Oscillation of Stars
BOVINE Binary Obscure Voters Network
SoHo SOUTH OF HOUSTON STREET
BREAD Brian Edwards Analytical Design
ABJ A big Giant
```

Output

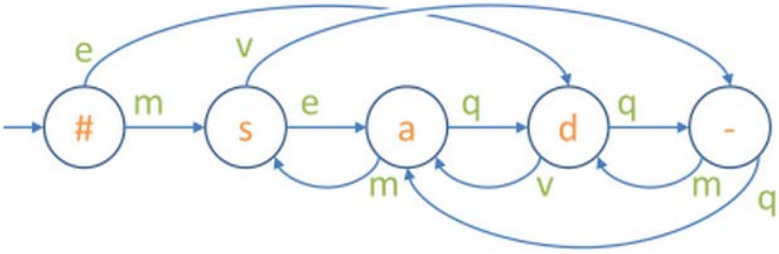
For each input line, the program must analyze the phrase and determine if the candidate is really an acronym for the phrase that follows, and if it is, what its properties are. The judges will expect the output to follow the patterns and wording shown in the examples below. The key words are complete vs. partial and simple vs. complex, or the use of the word not. Some acronyms could be parsed either simple or complex. In such cases the preferred analysis is simple.

```
SPBSG is a COMPLETE SIMPLE acronym for slowly pulsating B super-giants
GNU is a PARTIAL SIMPLE acronym for GNU is not Unix
COSTAR is a COMPLETE SIMPLE acronym for corrective optics space telescope axial
replacement
MOST is a PARTIAL COMPLEX acronym for Microvariability and Oscillation of Stars
BOVINE is NOT an acronym for Binary Obscure Voters Network
SoHo is a PARTIAL COMPLEX acronym for SOUTH OF HOUSTON STREET
BREAD is a COMPLETE COMPLEX acronym for Brian Edwards Analytical Design
ABJ is NOT an acronym for A big Giant
```

Problem 16 State Machine Decoder [11 points]

Summary

A finite state machine is a tool for modelling the behavior of a system. The concept is a machine with a finite number of states, where the machine can only be in one state at any given time. Additionally, the machine can change its state based on certain trigger conditions. State machines are useful for programming because it allows a program to follow a pre-determined series of actions based on a sequence of events.



State machines can be represented by diagrams to assist programmers in designing and understanding system behavior. Circles are used to represent states and arrows to represent transitions. States and transitions are distinguished by labels. An unlabeled arrow points to the initial (start) state of the system.

Of course, the diagrams are simply tools for communicating with organics. The machine follows the code regardless of what's in the diagram.

Write a program that can build a state machine and use it to decode a text message.

Input

The first line of input indicates the number of transitions. Each transition is listed on its own line and has three parts: a state label, a trigger condition, and a target state label. All parts are single characters and the initial condition is labelled with a hash tag. The last line of the input is a text string to be decoded.

```
10
# m s
# e d
- q a
- m d
a q d
a m s
d q -
d v a
s e a
s v -
meqqqmvmvq
```

Output

The program must decode the last line of the input according to the rules of the state machine. Each character of the text string should be interpreted as a trigger condition for a state change and the new state should be appended to the decoded string. The program must print the decoded string.

```
sad-as-dad
```

Problem 17 Spiral Triangles [12 points]

Summary

Spirals are beautiful patterns that can be generated using simple algorithms. For this program you'll generate spirals that follow a roughly triangular path as they wind clockwise out from the center, with horizontal motion going from right-to-left. The spiral will consist of regularly spaced vertices connected by edges as shown in the examples below. The edges should be drawn using the forward slash, backslash, and dash characters. Please note that in order to draw the spiral on a text console, one unit of horizontal motion uses three dash characters.

Input

Each line of input describes a single spiral. The first character indicates the direction of the first vertex away from the center. The second character indicates the number of segments, which is also the number of the final vertex that should be drawn. The points should be labelled sequentially from 0 through 9, then from A to Z. The last line of input contains two hash tags.

```
\ 2
- 5
/ D
# #
```

Output

The program must print the spirals described by the input. Remember the spiral must wind clockwise out from the center. Pay attention to the spacing between the vertices and edges and use periods to fill in empty spaces.

```
..0..
...\.
2---1

....3....
.../.\...
..2...4..
./.....\.
1---0...5

........A......
......./.\.....
......9...B....
...../.....\...
....8...1...C..
.../.../.\...\.
..7...0...2...D
./.........\...
6---5---4---3..
```

Problem 18 Downhill Mazes [12 points]

Summary

A downhill maze is a fun little navigation exercise. The maze is a grid of integers between 0 and 9. These numbers represent the height of the maze at each cell of the grid. The goal of the exercise is to move a heavy boulder along a path from the starting cell S (at height nine) to the target cell X (at height zero). You may roll the boulder to an adjacent cell to the north, south, east, or west (no diagonals), but only if the adjacent cell is not higher than the cell the boulder is in. In other words, you cannot roll the boulder uphill.

Write a program to find a path through a downhill maze.

Input

The first line of input indicates the number of rows and columns in the maze. The maximum for either number is nine. Every line after that is a single row in the maze.

```
5 9
3 3 3 3 2 1 S 8 9
3 1 1 3 3 0 6 8 7
1 2 2 4 3 2 5 9 7
1 2 1 5 4 3 4 4 6
1 1 X 6 4 4 5 5 5
```

Output

The program must print a grid of the same size as the input grid, but the output grid should print periods along the boulder's path. Every unvisited space should be marked with a hash "#" character. The start and target cells should keep their marks. There will only be one path through the maze.

```
. . . . # # S . #
. # # . . # # . .
. # # # . # # # .
. # # # . # # # .
. . X # . . . . .
```

Problem 19 Battleship [15 points]

Summary

The enemy is closing in and you need to make sure your fleet is ready for battle. To prepare, you need to confirm the ships in your fleet and ensure that they are sufficiently spread out. This will make it harder for your enemy to sink your fleet. The good news is that you have an overview of your fleet, but you don't know the exact number of ships under your command. Write a program to calculate how many ships of each type you have and verify that none of them are touching at their corners. Ships will not be touching on their sides.

Oh, one more thing. The ships in your fleet are more advanced than the traditional straight line ships of years past. In your fleet valid ship configurations include: Battleship, Carrier, Destroyer, Frigate, and Gunner. Below you'll find all of the valid ship shapes. Keep in mind that the ships can be facing up, down, left, or right.

```
Battleship:  Carrier:  Destroyer:  Frigate:  Gunner:
   X            --X       -X-         -X-       XX
   X            -XX       -X-         XXX       XX
   X            XX-       XXX         XXX       XX
   X            X--       XXX         -X-       XX
```

The first line of input contains a single integer S, indicating the size of the ship grid. The next S lines each contain S characters representing the ship grid. A dash is used to indicate water while an X denotes a piece of a ship. The maximum grid size is 20x20.

Your program needs to determine how many ships of each type are in the grid, and if any of the ships are touching at their corners. The first five lines of the output will list the number of Battleship, Carrier, Destroyer, Frigate, and Gunner ships in the grid. The last line will state how many ships are touching.

Input 1

```
15
-X---XX--------
-X----XX----X--
XXX----XX--XX--
XXX-------XX--X
----XX----X---X
XX-XXXX-------X
XX--XX----XX--X
XX--------XX---
XX---XX---XX---
---XXXX---XX---
-----XX--------
-X--------XX---
XXX--XXXX--XX--
XXX----------XX-
-X----XXXX-----
```

Output 1

```
3 Battleship
3 Carrier
2 Destroyer
2 Frigate
2 Gunner
0 ships are touching.
```

Input 2

```
20
-X-------------XX---
-X---XXXX--XX--XXXX-
-X---XXXX-XXXX-XX---
-X---------XX-------
----X------------X--
---XX------XXX---X--
--XX-------XXX---X--
--X----X----X----X--
------XXX---X-------
------XXX-----------
-XX----X-----XX-----
--XX----------XX-----
---XX----XX--XX-----
---------XX--XX-----
---XXXX--XX----XX---
--X------XX--XX-XX--
--X-----------XX-XX-
-XXX-----XX----XX---
-XXX----XXXX--------
---------XX---------
```

Output 2

```
3 Battleship
4 Carrier
3 Destroyer
3 Frigate
3 Gunner
5 ships are touching.
```

Input 3

```
8
XX--XX--
XX---XX-
XX----XX
XX------
--X--X--
--X--X--
-XXX-X--
-XXX-X--
```

Output 3

```
1 Battelship
1 Carrier
1 Destroyer
0 Frigate
1 Gunner
2 ships are touching.
```

Problem 20 Parenthetical List Object Notation [17 points]

Summary

Recently as I was drifting off to sleep I wondered, "What would the world be like if Netscape had chosen LISP for browser scripting instead of JavaScript?" I mean, you know, if they had hired programmers from AutoDesk it could have happened. Really. Maybe.

The answer, clearly, is that the web would be programmed in JavaLISP and apps would now be passing data using PLON -- Parenthetical List Object Notation. And the syntax rules would be something like this:

- At the top level, data is a represented as a list
- A list has zero or more elements enclosed by parentheses.
- A list element may be a word (a non-quoted alphabetic string) or a list
- There is a special list called a property list, which has these features:
  - the first element of the property list is the hash sign "#"
  - all remaining elements are lists that represent name/value pairs
  - the first element of a name/value pair must be a string
  - the second element of a name/value pair can be any valid element

A real PLON syntax, of course, would also allow for numbers and quoted strings, but for our contest this definition will do. Here's an example PLON string:

```
( # ( book ( # ( title MyBook ) ( authors ( MyName YourName ) ) ( date MMXIV ) ) ) )
```

Write a program to parse PLON data strings and query them using object dot notation. This notation uses property names separated by periods to query the object hierarchically and square brackets to indicate list item indices (starting from zero). For example, the expression `book.title` would return `MyBook`, and the expression `book.authors[0]` would return `MyName`.

Input

The first part of the input will consist of a PLON string, which may span multiple lines. You may assume that parenthesis, hash signs, and words are separated by one or more spaces and/or end-of-line characters. You may also assume the PLON string is valid (you don't need to validate the syntax). The next lines should be interpreted as object queries in dot notation. The last line will contain a single period.

```
( # ( book ( # ( title MyBook ) ( authors ( MyName YourName ) ) ( date MMXIV )
( characters ( ( Mary  Martha  Larry ) ( WuYan SuLiu ) ) )  (  formatList (
( # ( format paperback ) ( price XXIV ) ( pages CCCIC ) )   ( # ( format audioCD )
( price XIX ) )   ( # ( format hardback ) ( price XXXII ) ) ) ) ) ) )
book.date
book.authors[0]
book.characters[1][0]
book.formatList[2].price
.
```

Output

For each object query, the program must print a line with the object query followed by an equal sign and the correct property value. You may assume that each output value will be a single word and not a list.

```
book.date = MMXIV
book.authors[0] = MyName
book.characters[1][0] = WuYan
book.formatList[2].price = XXXII
```