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 newest CodeWars site – Conway, Arkansas. All you have to do is print "Arkansas, The Natural State!" to the screen.

Output

```
Arkansas, The Natural State!
```

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 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:

```
Wilfred
```

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, O Honorable Wilfred the Magnificent!  May I kiss your signet ring?
```

Problem 2

Running Out of Water

[2 points]

Summary

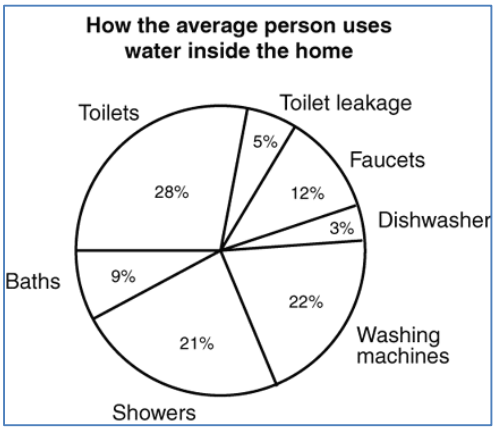
A small town in the California desert, Dry Gulch, has an underground water storage tank that contains 10,000 gallons of water. Thankfully (given that it's another drought season), the tank has not been discovered by any nearby golf courses, water parks, or mineral water companies, and is still dedicated for use only by the residents of Dry Gulch.

Given a weekly usage rate for the town residents, calculate the number of weeks the water will last.

Input

The input will consist of different weekly water usage rates ending with a zero.

```
1750
1000
4325
0
```



Output

For each line of input calculate the total number of full weeks that the water supply will last and print the output as follows.

```
1750 gallons per week will last 5 weeks
1000 gallons per week will last 10 weeks
4325 gallons per week will last 2 weeks
```

Problem 3 Queen Ann [3 points]

Summary

Queen Ann likes kittens, but she hates cats. She likes puppies, but she hates dogs. Queen Ann likes spoons, but not forks or knives. She likes summer, but she doesn't like heat or sunshine. She hates winter, but she likes freezing blizzards. Queen Ann likes pepper but not salt, and pizza but not pasta.

Without looking ahead, can you solve the riddle of what kinds of things Queen Ann likes or does not like?

** SPOILER ALERT **

Here is the solution to the riddle: Queen Ann only likes words that have double-letters, like her name. Write a program that can tell if a word is something that Queen Ann likes.

Input

The first line of input specifies how many words the program must read. Each word follows on a separate line. Words are composed of upper case English letters.

```
7
KITTENS
FORKS
WINTER
RIDDLES
TELEVISION
BOOKS
COWS
```

Output

For each input word, the program must print whether Queen Ann "likes" or "hates" the word.

```
likes KITTENS
hates FORKS
hates WINTER
likes RIDDLES
hates TELEVISION
likes BOOKS
hates COWS
```

Problem 4 Scientific Notation [3 points]

Summary

Scientists and engineers often deal with very large and very small things such as redwood trees and atomic particles. Our data values can similarly be very large or very small. In order to make it easier to read and write such numbers we use scientific notation.

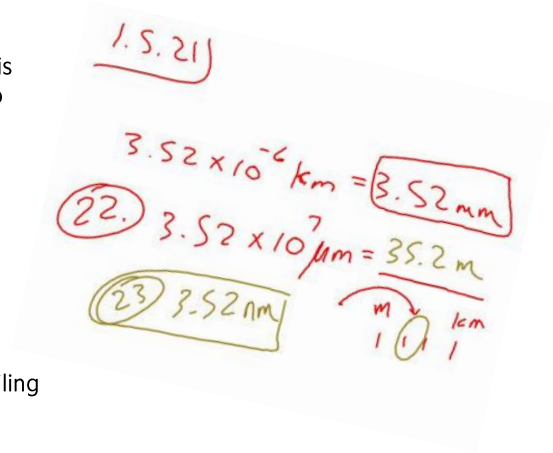
Here's an example of a number written in scientific notation:

```
3.926 x 10^4 (which is 39260)
```

Write a program that will read in a scientific number as a base number and a power of 10 and calculate the equivalent decimal value.

Input

Each line contains a pair of numbers, B and E, where B is the base number and E is the power of 10. B is greater than 1 but less than 10, and E is in the range of -10 to 10. The last line of input is two zeros.




```
4.296 3
3.8 -2
1.8 2
2.8678 1
0 0
```

Output

For each line of input calculate the actual value rounding to 2 decimal places. Trailing zeros to the right of the decimal point are required.

```
4296.00
0.04
180.00
28.68
```

Problem 5 Letter Tax [4 points]

Summary

Every year, local letter herders bring their flocks into town to pay their letter tax. The tax rate is variable based on other taxes paid by the herders. Don't get them started complaining about the cryptographic tax code or the irrational number tax -- there's no end to it.

Here's how the tax works: the tax officer gives the letter herder an integer number N. The letter herder must relinquish every Nth letter to the tax, starting with the first letter and separating out every Nth letter afterward.
Write a program to print the state of the flocks after the tax.

Input

The first line of input indicates the number of flocks the program will process. Every line after that begins with an integer (the tax rate N) and then is followed by a single space and a sequence of English letters. These are the representations of the flocks before the tax. As you probably already know, the maximum size of a letter flock is 64.

```
3
4 xCORxRECxT
5 agoodEbyeLDetteKrs
11 xTheQuickBrwownFoxJumpesOverTheLaxzyDog
```

Output

For each flock, the program must print the representation of the flock after the tax. The letters must be printed in the same order as the input, but without the taxed letters. The program must also print (on the same line) the number of letters remaining in the flock after the tax. The letters removed must start with the first letter in each flock and every Nth letter must be removed afterward.

```
CORRECT 7
goodbyeLetters 14
TheQuickBrownFoxJumpsOverTheLazyDog 35
```

Problem 6

Temperature Sensor

[5 points]

Summary

Hewlett Packard Enterprise customers operate datacenters with large numbers of compute, storage, and networking devices densely packed together. In these environments, the equipment could fail and possibly become damaged if allowed to overheat. So these devices are equipped with one or more temperature sensors that a program can read periodically to determine the device's temperature.

Temperature sensors must be calibrated because of slight variations in the manufacturing of the sensor hardware. So at 20°C one sensor might report a T value of 248 and another sensor might report a T value of 253 for the same temperature. So each sensor is calibrated at two temperatures, say 10°C and 20°C and the T values for these two temperatures are recorded into the sensor. The temperature sensor readings have a linear relationship to temperature. When a program reads a T value from the sensor it must also read the calibration values. The program must calculate the equation of the line through the two calibration points, and then using that equation it can compute the temperature in Celsius for the current T value.

Write a program to calculate the temperature of a sensor using two calibration values and the current sensor value.

Input

The first line of input indicates the number of sensors to read. Each line thereafter represents five integer values reported by a single sensor, in this order: T, T0, T1, C0, C1; where

- T is the current sensor's temperature value (T value)
- T0 is the sensor calibration value that corresponds to the 1st calibration temperature, C0
- T1 is the sensor calibration value that corresponds to the 2nd calibration temperature, C1
- C0 and C1 are the two calibration temperatures, given in units of degrees Celsius times eight
- T0 < T1, C0 < C1
- T may be less than T0, greater than T1, or anywhere in between.

```
3
450 350 550 160 240
270 300 600 150 250
640 280 480 170 220
```

Output

For each input line, the program must print the correct temperature in degrees Celsius. Remember that the C0 and C1 values are degrees Celsius times eight, so the program must divide by eight to arrive at the correct answer. Also keep in mind that the input values are all integers, but in order to calculate the correct answer the program may need to use floating point numbers. The answers must be correct to within +/- 1 degree Celsius.

```
25
17.5
32.5
```

Problem **7** **Share Your Letters** **[5 points]**

Summary

You may have heard that you learned everything you needed to know in kindergarten. If that's true, then why are you still in school? Does that seem right to you?

At CodeWars, we believe that you actually do learn quite a few useful things in school after kindergarten. For example, you probably learned to write programs sometime after kindergarten. However, one of the useful lessons we all should have learned, but we sometimes forget, is that it's nice to share your things with others. With that in mind, your task for this problem is to write a program that can determine the set of letters shared in common among three words.

Input

The first line of input indicates the number of word triplets the program must read. Each line after contains three words, and we use that term loosely, separated by one or more spaces. Don't assume there will only be one space between words.

```
3
TEST         MEANT           TIME
KINDERGARTEN CHICKENFEATHERS SPECIALITIES
ABSURD       SUBORDINATE     DUMBELLS3
```

Output

For each triplet, the program must print all the letters that are shared by all three words. If a letter appears multiple times in each word, then it should also appear the corresponding number of times in the output. The letters must be printed in alphabetic order.

```
ET
AEEIT
BDSU
```

Problem 8 Parallelogram Words [5 points]

Summary

In the early days of computing we had dot matrix printers that would output everything in a fixed width font. Those of us who were programming back then created all kinds of neat word art. Today you may have seen ASCII pictures or even ASCII animations. Let's create our own word art!

Input

The first line of input contains the number of words that follow. Each following line will contain one word up to 10 characters long. All letters will be upper case.

```
3
TEST
SAMPLE
ART
```

Output

Print each given word horizontally once and multiple times vertically so that each letter in the horizontal word matches the position of that letter in the vertical words.

The horizontal word will be in the middle of the output. The first vertical word uses the first letter of the horizontal word. The last vertical word uses the last letter of the horizontal word.

Each parallelogram should be separated from the next by a blank line.

```
   T
  TE
 TES
TEST
EST
ST
T

     S
    SA
   SAM
  SAMP
 SAMPL
SAMPLE
AMPLE
MPLE
PLE
LE
E

  A
 AR
ART
RT
T
```

Problem 9 Perfect Painting [6 points]

Summary

Consider a block of wood that has dimensions of L, W, D (all integers, in centimeters). Now paint that block on all faces and let it dry. Finally, cut the block into 1 cm cubes. Let's say the block is:

- "PERFECT" if the number of cubes with paint is identical to the number of cubes without paint.
- "MORE than Perfect" if there are more painted cubes than not painted.
- "LESS than Perfect" if there are less painted cubes than not painted.

Write a program that, given the Length, Width, and Depth of a block, outputs the classification of the block.

Input

Each line of input has three integers: the Length, Width and Depth of the block. The input ends with three zeroes.

```
5 6 7
10 11 12
8 10 12
0 0 0
```

Output

The program must print the classification for each block. Capitalize only MORE, LESS, or PERFECT as appropriate.

```
A 5x6x7 block is MORE than Perfect.
A 10x11x12 block is LESS than Perfect.
A 8x10x12 block is PERFECT.
```

Problem 10 Tic-Tac-Toe [6 points]

Summary

Write a program that will read in a Tic-Tac-Toe board configuration and determine if there was a winner. In our data representation the two players will be represented by X and O, respectively. An '=' indicates that the game ended before a move in this position was necessary.

The game of tic-tac-toe begins with an empty game board that looks like this:

```
__|__|__
__|__|__
  |  |
```

The starting player then places an X into any board position:

```
_X_|___|___
___|___|___
   |   |
```

The second player will place an O into another board position. The players then alternate moves until one player manages to place all X's or all O's along a row, column, or diagonal. Here are some example games. The first three board configurations show wins for O, X, and X respectively. The fourth board configuration shows a draw.

```
=XX    XO=    XOX    XOX
OOO    OX=    =OX    OOX
XOX    XOX    O=X    XXO
```

Input

Each line of input contains a complete board configuration of 9 characters. The first three characters represent the top row, the next three characters represent the middle row, and the last three characters represent the bottom row. The end of the input will consist of a board filled with all '='.

Note: The board configurations for this problem will never have a pair of winning directions.

```
=XXOOOXOX
XO=OX=XOX
XOXOOXXXO
=========
```

Output

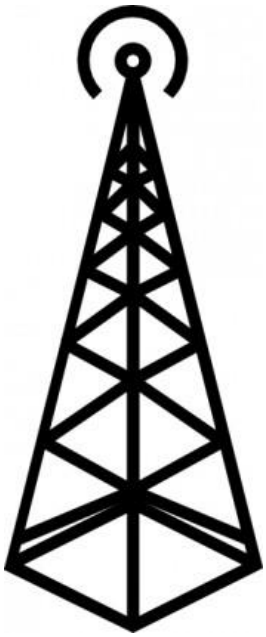
The output should first print which player won followed by the board. The winning moves in the board should be marked by replacing the X's or O's with '$'.

```
Player O won.
=XX
$$$
XOX

Player X won.
$O=
O$=
XO$

There was a tie.
XOX
OOX
XXO
```

Problem 11 Location Awareness [7 points]

Summary

Many modern computing devices have Location Awareness features. Phones, tracking systems, and self-driving vehicles have the ability to determine their current location. Often this is accomplished using a trilateration algorithm. If the device can receive signals from three sources whose locations are known, then it can determine its location from that data.



For this problem, you will write a trilateration algorithm (explained below) for an autonomous robot using signals from three towers positioned around a square arena. The arena is an integer grid with walls at the four lines defined by x=100, y=100, x=-100, and y=-100. The robot may be positioned anywhere within the arena. Tower 1 is located at (x,y) position (0,100), tower 2 at (-100,-100), and tower 3 at (100,-100).

Here's how the system works: the towers broadcast distinct signals that the robot can receive. The towers are all powered by one common battery. When the robot is near a tower, the signal strength is high, but the farther the robot is from the tower, the weaker the signal. The strength of a tower's signal is given by the following equation:

```
s = P / ( d * d )
```

The variable P is the transmission power and d is the distance from the tower to the robot. When the battery is fully charged the signal is very strong. But over time, as the battery's energy is used, the signal power is reduced. So P is the same for each tower, but it changes over time. Also, the robot has no direct way to measure P. So it is not possible to make an exact calculation for the distance to a tower using the signal strength. You'll have to think of how to use all three signals to solve the problem. The only math you need to know is that the distance between two points (x0,y0) and (x1,y1) is given by this equation:

$$d = \mathrm{sqrt}((x0-x1)^2 + (y0-y1)^2)$$

Use the signals from the three towers to determine the robot's location on the grid.

Input

Each line of input has three floating-point numbers separated by one or more spaces. These numbers are the signal strengths from towers 1, 2, and 3, in that order, for each location of the robot. The input ends with three zeros.

```
5.432   2.716  2.716
6.733   0.956  1.284
501.345 2.102  1.878
2.207   2.644  662.852
0 0 0
```

Output

For each input line, the program must print the exact integer x and y location of the robot.

```
0 0
21 35
-14 99
93 -90
```

Problem 12 Linear E [7 points]

Summary

Linear A is the name of a writing system used in ancient times on the island of Crete and the surrounding region. Although Linear A was discovered in the late 1800's, the language has not yet been deciphered and remains a mystery. Interestingly, some of the examples of Linear A were carved into round discs with the writing following a spiral path. Archaeologists do not yet know if the writing should be read from center outward or from the edge inward. The spiral form of Linear A has inspired this CodeWars problem.

Write a program to read a grid of English letters recorded in a spiral sequence and print the text left-to-right. For this program you should assume the text begins at (or near) the center and spirals out in a clockwise direction.

Input

The first line of input indicates the number of rows and columns of the letter grid. The grid itself begins on the next line. There will be a region in the center of the grid with open spaces that will allow the program to determine where the spiral begins. There may also be blank spaces along the edge of the grid. All spaces are important!

```
7 9
 AreFunTo
 sarsProR
 tW    be
ene    la
vedoC  ed
lmetatSmA
oSoTnuFdn
```

Output

The program must locate the starting position of the text and print the entire message in standard left-to-right sequence.

```
CodeWarsProblemStatementsAreFunToReadAndFunToSolve
```

Problem 13 Star Catalog [8 points]

Summary

Space: the final frontier. Imagine a future in which humans have learned to manipulate Higgs boson fields using jump field effect (JFE) generators powered by nuclear fusion. Humans exploring the galaxy! It's an awesome idea, right? And someone has to write the astronavigation software. It might as well be you. But there are a few things you'll need to know.

You have access to a catalog of star systems near our home star, Sol. For historical reasons, star coordinates are listed in the catalog by right ascension (RA), declination (dec), and distance from Sol in light-years (LY). RA and dec are roughly analogous to longitude and latitude. But we need to calculate distances between any two stars in the catalog. It would be much easier if our stars were listed with Cartesian (x,y,z) coordinates. That conversion will require some attention to detail.

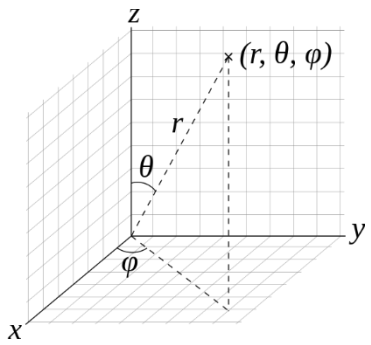
The celestial equivalent of latitude is declination and is measured in degrees North (positive numbers) or South (negative numbers) of the Celestial Equator. One degree of declination is divided into 60 minutes. The celestial equivalent of longitude is right ascension. Right ascension can be measured in degrees, but for historical reasons it is more common to measure it in time (hours, minutes, seconds): the sky turns 360 degrees in 24 hours and therefore it must turn 15 degrees every hour; thus, 1 hour of right ascension is equivalent to 15 degrees of (apparent) sky rotation. One hour of right ascension is divided into 60 minutes.

Mathematicians and physicists sometimes use a spherical coordinate system to describe locations in three-dimensional space using two angles and a distance from a fixed origin. This is similar to astronomical coordinates, but in spherical coordinates the polar angle is measured from the positive Z-axis (North).

IMPORTANT NOTES: The conversion process involves three steps:
(1) Convert RA (hours+minutes) and dec (degrees+minutes) to real (floating-point) values in degrees. Pay close attention to the different meanings of the word "minute" in RA and dec. Also be careful when converting negative declination values. For example, the decimal equivalent of "-16 30" is -16.5 degrees.

(2) Convert (RA,dec,LY) to spherical coordinates (r,θ,φ) in radians.

```
r = LY
θ = (90 - dec) * π / 180
φ = RA * π / 180
```



(3) Convert (ρ,θ,φ) to (x,y,z) using trigonometric transformations

```
x = r * sin( θ ) * cos( φ )
y = r * sin( θ ) * sin( φ )
z = r * cos( θ )
```

Input

The first line of input indicates the number of stars in the catalog. Each star is described on its own line with the following fields: the star name, RA hours, RA minutes, dec degrees, dec minutes, star classification, absolute magnitude, and the distance from Sol in light-years. Each field is separated by one or more spaces. There may be up to one hundred stars in the catalog.

```
4
Sol                   00 00.0 +00 00  G2V                  4.83  0.00
Alpha-Centauri        14 39.6 -60 50  G2V+K1V+M5.5V        4.06  4.39
Sirius                06 45.1 -16 42  A1V+DA2              1.43  8.60
Teegarden's-star      02 53.0 +16 53  M6                  17.22 12.51
```

Output

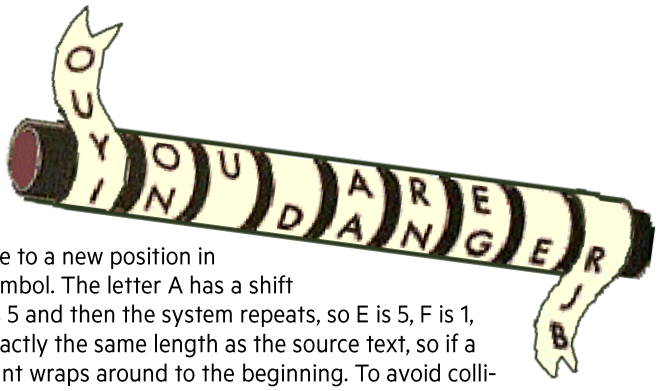
For each star, the program must print the star's name and (x,y,z) coordinates. Values must be accurate to within 1/10 of a light-year.

```
Sol x=0.00, y=0.00, z=0.00
Alpha-Centauri x=-1.64, y=-1.37, z=-3.83
Sirius x=-1.61, y=8.08, z=-2.47
Teegarden's-star x=8.72, y=8.20, z=3.63
```

Problem 14 Variable Shift Encoding [9 points]

Summary

Codes and ciphers have been used since ancient times. In 405 BC, the Greek general Lysander of Sparta was sent a coded message written on the inside of a servant's belt. When Lysander wound the belt around a wooden baton the message was revealed: Persia was about to go to war against him. He immediately set sail and defeated the Persians.



For this program you'll use a cipher that shifts each symbol in a message to a new position in the message. The number of places to shift depends on the previous symbol. The letter A has a shift value of 1, B has a value of 2, etc. The maximum shift value for a letter is 5 and then the system repeats, so E is 5, F is 1, G is 2 and so on. A space character has a shift of 6. The cipher text is exactly the same length as the source text, so if a letter is shifted past the end of the cipher text length, then the shift count wraps around to the beginning. To avoid collisions, shift counting skips over characters that have already been placed into the cipher string. Finally, lines of text are broken into segments by periods, so each sentence is encoded separately from other sentences.

For example, consider the following source text: FIRST TEST. The F is assigned as the first character of the cipher text (there is no previous character). Then the next character I is placed 1 position after F (because F has the shift value of 1), so the cipher text becomes FI--------, where dashes represent characters in the cipher that haven't yet been filled. Then the third character R is placed 4 positions after the I (because I has the shift value of 4), so the cipher text becomes FI---R----. Then S is placed 3 positions after R, resulting in FI---R--S-, and T is placed 4 positions after S, resulting in FI--TR--S-. The completed cipher text is

```
FIE TRSTST.
```

Write a program to decode cipher texts based on the encoding scheme described here.

Input

The first line of input indicates how many lines of cipher text the program must read. Each cipher will contain one or more sentences that end with a period. The total line length (including periods) will be less than or equal to eighty characters. Sentences will only contain upper-case English letters and spaces. Notice that all spaces are important and there may be two or more adjacent spaces in the cipher text.

```
4
FIE TRSTST.
TT OTHA IENESIGT RNSLLSCI EE ELN.
WIDEONKE ETCA L .  AI  IUSTTNF L FOO. NT  HCCIOSEDUM WEA.
HNOERBTE EUS SCI A.  RU IDIUIOLBSENDRCTO.
```

Output

The program must print the decoded text in the same line and sentence sequence of the input cipher text.

```
FIRST TEST.
THIS SENTENCE IS A LITTLE LONGER.
WE LIKE TO DANCE. IT IS A LOT OF FUN. COME DANCE WITH US.
HE CANT BE SERIOUS. DONT BE IRRIDICULOUS.
```

Problem 15 Cube Rotation [9 points]

Summary

This year we're dealing with an actual cube, not cubicles. Our cube is a mini version of the famous Rubik's Cube. The original puzzle was called the Magic Cube and was invented in 1974 by Hungarian professor Erno Rubik. We will work with a 2x2x2 cube that uses the same colors as the original: Green, Red, Blue, Orange, Yellow, and White.

You will write a program that will scramble a solved cube. The moves used to scramble it will be the input to your program. The output will be the front face of the cube after each move.

The cube has six sides and here are the starting colors for each face of the cube:

```
Front:  Green
Left:   Orange
Right:  Red
Back:   Blue
Up:     White
Down:   Yellow
```

A move is considered a 90 degree clockwise rotation of one face of the cube.

Input

The input will consist of a series of move instructions to be performed on the solved cube. Each move is one 90 degree clockwise rotation of one face of the cube: Front, Back, Left, Right, Up, and Down. There will be one move instruction per line. The end of the input is a single period.

```
U
R
R
D
D
L
U
U
.
```

Output

The output will display the front face of the cube after each move. Starting with the initial configuration, print the move followed by the front face of the cube. Use the starting letter of each color to represent a tile.

```
Start
G G
G G

U
R R
G G

R
R Y
G Y

R
R B
G O

D
R B
O O

D
R B
R B

L
W B
W B

U
R R
W B

U
G W
W B
```

Problem 16 Micecraft Gauntlet [10 points]

Summary

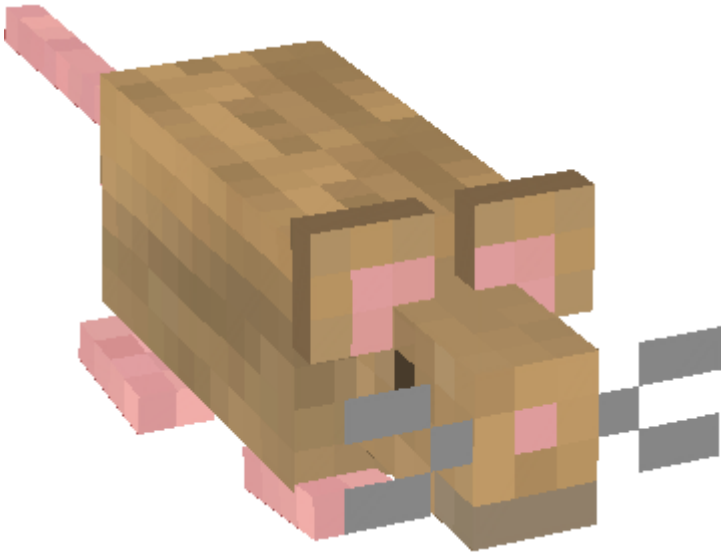
Each month, the gamer RatMaster creates a set of challenges for all Micecraft players. He calls the competition "The Gauntlet" and it is a highly anticipated event. At 9:15 a.m. on a Saturday, he opens his arena with many different challenges. They range in difficulty from simply "design a Muenster Tree" to "Avoid all RatTraps while collecting 100 BrieCoins." To add to the difficulty, the Gauntlet closes at precisely 12:15 p.m. (One of RatMaster's ongoing criticisms is "How can anyone hope to complete all these challenges in 3 hours!?")

The challenges are numbered starting from 1, and each is worth a certain number of points. They may be completed in any order, and anyone who completes a challenge is awarded its points. Challenge 1 is worth the least and the highest numbered problem is worth the most, and each challenge's point value is never less than the previous one. For example, if there were 5 challenges, they could be worth (1 2 4 5 8) or (1 1 1 2 3) or (3 4 4 8 20). But each challenge's point value is kept secret until the end of the Gauntlet! 90 minutes into the competition, RatMaster posts the current standings without any competitor names, showing only their total number of points and which problems each has solved. Everyone looks forward to this reveal, because with a little guesswork, they can determine the point-values for most challenges and choose which remaining ones to attack to reach their best scores.
You must write a program to interpret the scoreboard and determine the number of points each challenge is worth.

Input

Each line includes the total points for one competitor, a space, then a list of 2-digit challenge numbers in increasing order separated by spaces, ending with '00'. The list ends with a single 0.

```
Example 1:
6 01 02 03 00
7 03 04 00
5 01 06 00
0

Example 2:
3 01 04 00
15 04 09 00
0

Example 3:
13 02 07 00
35 01 02 03 04 05 06 15 00
13 01 06 00
6 02 05 00
5 03 04 00
0
```



Output

The program must print the point value for each challenge that can be accurately determined, beginning with challenge 1. If a value cannot be determined, it should print a question mark "?". There should be an entry (points or ?) from 1 to the highest challenge number found on the scoreboard.

```
Output 1:
1 2 3 4 4 4
        ^-- Problem 5 must be worth 4 points since problems 4 and 6 are.

Output 2:
1 ? ? 2 ? ? ? ? 13

Output 3:
2 2 2 3 4 11 11 11 11 11 11 11 11 11 11
```

Problem 17 All Your Nets Are Belong To Us [12 points]

Summary

In a computer network, the network switches need to be configured for routing, access control, quality of service, and many other operating parameters. For example, a company's sales traffic might be given higher priority than, say, YouTube videos. In the early days of the internet, network administrators configured each switch one-by-one. Not only was this boring, but as the internet grew and the rate of change increased, configuring switches became a large problem. Now, Hewlett Packard Enterprise and other companies make network switches that can be configured using software that manages the entire network. This approach is called Software Defined Networking (SDN), because you can't really sell a product with a feature called All Your Nets Are Belong To Us*.

For this program you will simulate a simple SDN system. The program must connect network nodes using either single (low bandwidth) or double (high bandwidth) connections.

Input

The first line of input is the number of rows and columns in the square input grid (maximum value is ten). Network nodes are represented by single-digit numbers, while empty cells contain periods. Two examples are shown below.

```
5
3 . 4 . 2
. . . . .
. . 2 . .
. . . . .
2 . . . 1
```

```
7
2 . 4 . 3 . 4
. . . . . . .
. . . 3 . 2 .
. . . . . . .
. . . 2 . . 2
1 . . . . . .
. . 3 . . 2 .
```

Output

The program must connect all the nodes with straight lines that do not overlap other lines and print a diagram of the network configuration. Connections may be single or double. For each node, the number of connections must match the integer value for that node. Use - and = to draw single and double horizontal lines, and use | and " to draw single and double vertical lines. The network grids used in this contest only have one valid solution.

```
3 - 4 - 2
" . " . |
" . 2 . |
" . . . |
2 . . . 1
```

```
2 - 4 - 3 = 4
| . " . . . "
| . " 3 - 2 "
| . " " . | "
| . " 2 . | 2
1 . " . . | .
. . 3 - - 2 .
```

* *For those too young to remember, "All Your Base Are Belong to Us" was a poorly translated line of dialog from a 1991 video game called Zero Wing. The phrase became a popular internet meme in 2000 that took on a life of its own for a few years.*

Problem 18 Hexagonally Amazing [12 points]

Summary

This problem is all code, no tinsel. Solve a maze set within an ASCII hexagon grid. Print the solution. That is all.
We'll represent hexagon maze intersections in ASCII text with the letters, the # symbol, the @ symbol, and the question mark. Hexagons are connected by dashes and slashes, which represent paths through the maze. Every other line contains N-1 hexagons, where N is the number of hexagons in the first line. For example, a fully connected ASCII three by five hexagon grid looks like this:

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

Input

The first line of input contains the number of rows and columns in the maze. The remaining lines are the maze itself. The maze entrance will be marked with an @ symbol, and the exit with a question mark. There will not be any loops in the maze; in other words, there is never more than one path from any hexagon to any other hexagon. The line endings may (or may not) have extra spaces, so don't depend on input lines being padded with spaces at the end.

```
7 8
#---#---#    #---#    #---#    #
      \           / \    \      /
  #---#---#    #    #    #---#
 /           \      /    /      \
@---#---#    #---#    #    #---#
    /      \           /    /
  #    #---#---#    #    #    #
 / \                /    / \ /
#    #---#---#---#    #    #    #
 \             \         \      /
  #    #    #    #    #    #---#
   \ / \    \    \    \ /
#---#    #---#    #    ?---#---#
```

Output

The program must print the maze with the path highlighted using capital letters. The letters must mark the path in alphabetic order, starting with A as the first hexagon after the @ symbol.

```
#---#---#    #---#    #---#    #
      \           / \    \      /
  #---#---#    #    #    I---J
 /           \      /    /      \
@---A---#    #---#    H    L---K
    /      \           /    /
  B    #---#---#    G    M    #
 / \                /    / \ /
#    C---D---E---F    N    #    #
 \             \         \      /
  #    #    #    #    #    O---#
   \ / \    \    \    \ /
#---#    #---#    #    ?---#---#
```

Problem 19 Craft Master [13 points]

Summary

Many computer games allow players to craft items in-game by combining raw materials using a crafting interface. The list of materials used to craft an item is often called a recipe and the materials are called ingredients. Also, items that have been crafted may sometimes be used as crafting ingredients for other items.

If all of this sounds like it could get confusing, well, yeah, it can. Players who want to craft the Uber Awesome Hammer of Poundation may find themselves lost in the details of how many Iron Icicles they actually need to collect before the thing is finally made. That's where you can help.

Write a program to print a list of all the raw materials required to craft an item.

Input

The input file will consist of two sections. The first line is the number of recipes, followed by one recipe per line. Each recipe consists of an item name, the number of ingredients, followed by the number and name of each ingredient. Recipes will have two to four ingredients. After the recipes, each line will contain the name of an item to be crafted. All names are in CamelCase, so there are no spaces in the names. The input ends with the word GO.

```
9
UberHammerOfPoundation 3 1 UberHammerHead 1 UberShaft 1 UberHandle
UberShaft 3 3 IronIcicle 1 IridiumBar 1 MarkOfUber
UberHammerHead 3 1 MarkOfUber 4 IronIcicle 1 IridiumBall
UberShovelBlade 3 1 MarkOfUber 1 IridiumBar 4 IronIcicle
IronIcicle 2 5 IronLump 15 IceCrystal
MarkOfUber 3 7 PlatinumFlake 15 MagnesiumDust 1 IridiumNugget
IridiumBar 2 10 IridiumNugget 1 PlatinumFlake
IridiumBall 2 8 IridiumNugget 1 IronLump
UberHandle 3 1 RawhideStrap 1 BlackDye 1 RollDuctTape
UberHammerOfPoundation
UberShovelBlade
GO
```

Output

The program must print the name of each item the player wants to craft enclosed in square brackets, followed by the name and quantity of all raw materials. Any ingredient that has no recipe is a raw material. Each raw material used in the final item must be printed exactly once with the total quantity required for the final item. The raw materials must be printed in alphabetic order.

```
[UberHammerOfPoundation]
BlackDye 1
IceCrystal 105
IridiumNugget 20
IronLump 36
MagnesiumDust 30
PlatinumFlake 15
RawhideStrap 1
RollDuctTape 1
[UberShovelBlade]
IceCrystal 60
IridiumNugget 11
IronLump 20
MagnesiumDust 15
PlatinumFlake 8
```

Problem 20 Enigami Machine [16 points]

Summary

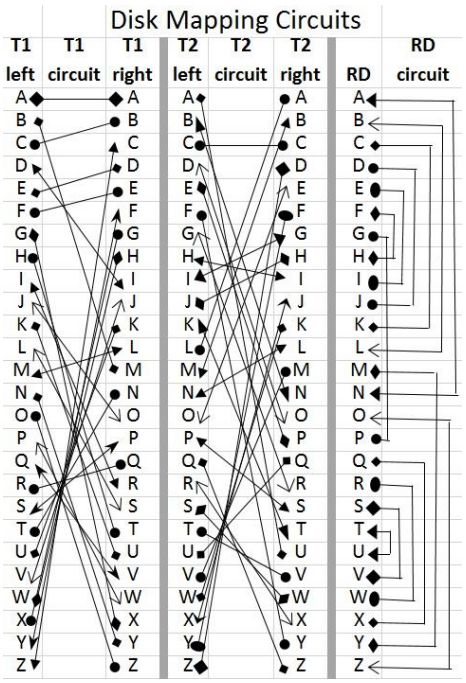
During World War II, messages were encrypted with an Enigma Machine. Through the dedicated effort of skilled cryptographers (and the predictability of some encoded messages), the Enigma Machine was deciphered, helping to end the war early.

Our rivals have created their own machine to encrypt their messages based on the Enigma and called it Enigami. Enigami is not the ancient Japanese art of sentence folding; it's just the word "imagine" spelled backward. We have analyzed the machine and need you to decode their messages.

The Enigami Machine consists of a series of disks: The Input disk (ID), two Translation disks (T1, T2), and a Reflector disk (RD). To encrypt one letter, the user presses its key, causing an electrical signal to pass left to right through ID, T1, and T2; the signal is reflected back to a different connector at RD, returning through T2, T1, and ID, where the signal turns on a light for the letter corresponding to the encrypted value. After each letter, T1, T2 and RD rotate, resulting in a different encryption circuit path for the next letter.

All disks are labeled sequentially A-Z. There are no spaces, digits, or punctuation. T1, T2, and RD connect signals between letters in this way (spaces are added here only for readability):


```
Input (left): ABCDE FGHIJ KLMNO PQRST UVWXYZ
  T1 (right): AMBID EXTRO USLYZ WVQPN KJHGFC
  T2 (right): UNCOP YRIGH TABLE SZXWV QMKJFD
  RD  (left): NLKJI HPFED CBYAZ GXWVU TSRQMO
```



T1 and T2 translate the signals left-to-right across the disk. RD connects letters on the left side of the disk in pairs.

Rotation

The ID does not move. The remaining disks all rotate. After a letter is encrypted, the disks rotate simultaneously:

- T1 advances one position, always.
- T2 advances one position if T1 starts at position A or O.
- RD advances one position if T2 starts at position A or O.

Encryption

- The initial disk configuration is the Message Key for the encryption. With a Message Key of "COD", the initial disk positions are T1=C, T2=O, RD=D.
- After encoding the first letter, T1 rotates to "D". T2 doesn't rotate (stays "O"). RD rotates to "E".
- After encoding the second letter, T1 rotates to "E". T2 doesn't rotate (stays "O"). RD rotates to "F".
- Neighboring disks connect to each other letter-to-letter.
- With the Message Key of "COD", the final encoding of the word "CAT" is "BWJ" (see figure on next page)

Decryption

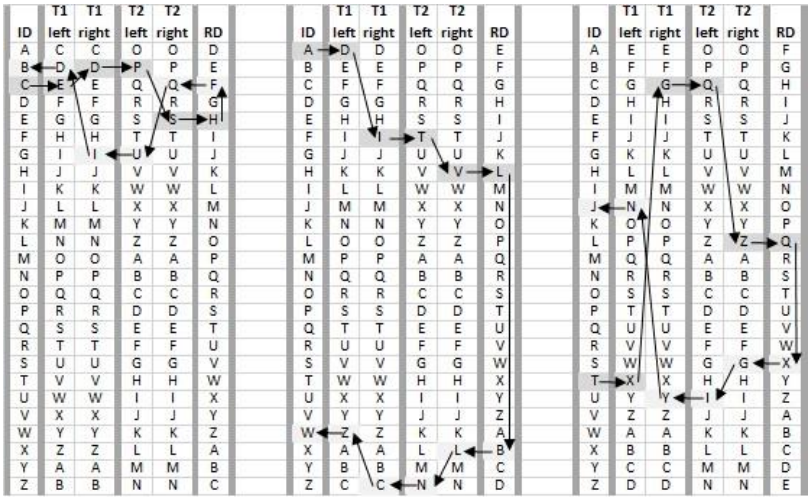
Because the original and encrypted letters are the endpoints of a single circuit path, the encrypted letters can be decrypted with exactly the same process. Starting with the Message Key, run the encrypted message through the Enigami Machine to find the original message.

Problem 20 Enigami Machine (continued) [16 points]

Your Task

Each day, our rivals use a new 3-letter Daily Key, and every message gets its own 3-letter Message Key. The complete encrypted communication consists of first the encrypted Message Key (using the Daily Key), then the encrypted message (using the Message Key). Here are the steps:

1) Set the rotors to match the Daily Key (i.e. "SPY").
2) Hit the three characters of the Message Key (i.e. "COD"), and transmit these three encrypted characters ("NEH").
3) Set the rotors to match the Message Key ("COD").
4) Encode and transmit the message. ("CAT FOOD IS AWESOME!" becomes "BWJ YBYN QI BBUNUGO!".)



Encryption of CAT to BWJ

The final encrypted text that is sent looks like this: **NEH BWJ YBYN QI BBUNUGO!** Here, "NEH" is the encrypted Message Key "COD" (encrypted using the Daily Key "SPY"), and the rest is the encrypted message (encrypted using the Message Key "COD").

Luckily, we receive transmissions from one foolish Enigami operator who is overcautious (and untrained in cryptography.) He types the Message Key twice before encrypting the message. He thinks he's being helpful, because the recipients can validate the Message Key since they'll receive it twice. His first two steps look like this:

1) Set rotors to match the Daily Key (i.e. "SPY").
2) Hit the three characters of Message Key (i.e. "COD"), hit the same three characters and transmit these six encrypted characters ("NEHNIW").

So his encrypted text looks like this: **NEHNIW BWJ YBYN QI BBUNUGO!** Since we know his message begins with two copies of the Message Key, you should be able to decrypt it (perhaps with a search through all possible Daily Keys.) This will tell us the Daily Key for any communications they may send that day! And of course, with a decrypted Message Key, you can also decrypt his message!

Input

Each line includes the foolish communication from one day. The first 6 characters are the encrypted Message Key (repeated). Then the encrypted message follows. Each line uses a different Daily Key. The last line is a single period.

```
NEHNIW BWJ YBYN QI BBUNUGO!
KSTFPD URT DFL ATRDNGPN QY TVDON?
.
```

Output

For each line, your program must determine (1) the Daily Key, (2) the Message Key, and (3) the decrypted message. Print all on the same line as below. Since spaces, punctuation and digits cannot be encrypted, copy them directly to the output message.

```
DK:SPY    MK:COD    MSG: CAT FOOD IS AWESOME!
DK:HEY    MK:YOU    MSG: DID YOU REMEMBER TO FLOSS?
```

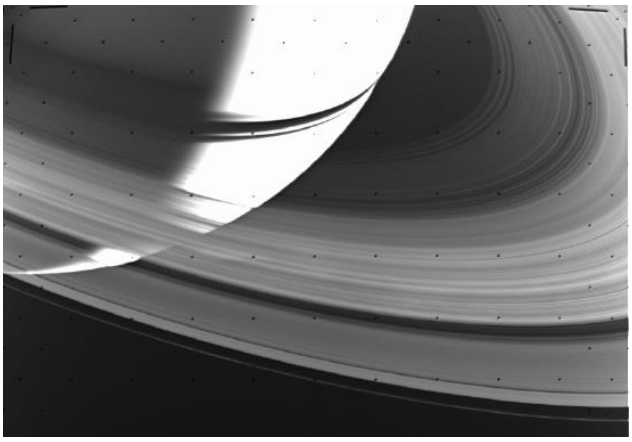
Trivia: This "foolish operator" scenario was actually one of the original flaws in the use of the Enigma machine. Read the paragraphs about September 1938 at https://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma

Problem 21 Astro-Navigation [18 points]

Summary

Space: the final frontier (again). Imagine a future in which humans have learned to manipulate Higgs boson fields using jump field effect (JFE) generators powered by nuclear fusion. Humans exploring the galaxy! It's an awesome idea, right? And someone has to write the astronavigation software. It might as well be you. But there are a few things you'll need to know.

First, the distance a starship can jump depends on the size of the JFE and the starship mass. Each starship is rated for the maximum distance it can jump, in light-years. Therefore long journeys must often be broken into a sequence of smaller jumps. This brings us to the second point: stars are not uniformly distributed in the galaxy. There may be gaps between the origination and destination too large for a ship's JFE to cross. In such cases, the optimal path may include motion away from the destination to circumvent such gaps. Keep in mind that the ship must refuel at each stop, so ending a jump in the middle of empty space is sub-optimal.



Finally, you have access to a catalog of star systems with refueling stations. For historical reasons, star coordinates are listed in the catalog by right ascension (RA), declination (dec), and distance from Sol in light-years (LY). These have been converted to (x,y,z) Cartesian coordinates by other team members (see the Star Catalog problem).

The challenge here is to find the shortest flight plan for a specific starship to travel from the origination to the destination. You can calculate the distance between two stars s1 and s2 using this formula:

$$d = \mathrm{sqrt}[\ (x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2\ ]$$

Then strap into your safety harness and get ready to jump!

Input

The input is arranged into two sections. The first line of section one indicates the number of stars in the catalog. Each star is described on its own line with the following fields: the star name, classification, absolute magnitude, and x, y, and z coordinates in light-years. Each field is separated by one or more spaces. There may be up to one hundred stars in the catalog. The first line of section two indicates the number of journeys the program must navigate. Each journey is described by a separate line with the origination, the destination, and the maximum number of light-years that the starship is capable of jumping before refueling.

```
10
Sol                     G2V                 4.83    0.000    0.000    0.000
Alpha-Centauri          G2V+K1V+M5.5V       4.06   -1.641   -1.372   -3.833
Luhman-16               L8+T1              14.20   -3.751    1.195   -5.285
Sirius                  A1V+DA2             1.43   -1.611    8.078   -2.471
Epsilon-Eridani         K2V                 6.18    6.201    8.296   -1.724
Groombridge-34          M1+M3              10.32    8.329    0.670    8.074
Epsilon-Indi            K4V+T1+T6           6.89    5.660   -3.157   -9.897
Tau-Ceti                G8V                 5.68   10.283    5.021   -3.267
Teegarden's-star        M6                 17.22    8.719    8.202    3.633
Kapteyn's-Star          M1                 10.87    1.890    8.834   -9.039
3
Sol                     Alpha-Centauri                  5
Teegarden's-star        Sirius                          5
Groombridge-34          Alpha-Centauri                  9
```

Output

For each journey, the program must print several pieces of information. First, it must print the names of the origination and destination. Then it must also print the straight-line distance from the origination to the destination and the number of jumps needed for the journey. Next it must print each jump of the journey, giving the departure, arrival, and distance jumped. Finally, it must print the total distance travelled. There must be a blank line between each journey. The program won't be judged on strict output formatting, but it's best to follow the example output and keep the output data clear and easy to find. If the judges can't find the data, they will fail the program. Distances must be accurate to within 1/10 of a light-year.

If the program cannot find a valid path, it must print a message like, "no route from origination to destination".

```
JOURNEY from Sol to Alpha-Centauri, max jump: 5 LY
STRAIGHT LINE DISTANCE: 4.39 LY
number of jumps: 1
Sol to Alpha-Centauri; 4.39 LY
Total Distance: 4.39 LY

JOURNEY from Teegarden's-star to Sirius, max jump: 5 LY
STRAIGHT LINE DISTANCE: 12.00 LY
No route from Teegarden's-star to Sirius

JOURNEY from Groombridge-34 to Alpha-Centauri, max jump: 9 LY
STRAIGHT LINE DISTANCE: 15.66 LY
number of jumps: 5
Groombridge-34 to Teegarden's-star; 8.75 LY
Teegarden's-star to Epsilon-Eridani; 5.92 LY
Epsilon-Eridani to Sirius; 7.85 LY
Sirius to Luhman-16; 7.74 LY
Luhman-16 to Alpha-Centauri; 3.63 LY
Total Distance: 33.89 LY
```