Problem A : Secret Party *(blue balloon)*

At the end of this **First Edition** of computer engineering student's day, organize by CERADEI-GIT, a secret contest party is going to take place. The organization team invented a special mode of choosing those participants that were to assist with washing the dirty dishes. The contestants would line up in a queue, one behind the other. Each contestant got a number starting with 2 for the first one, 3 for the second one, 4 for the third one, and so on, consecutively.

The first contestant in the queue was asked for his number (which was 2). He was freed from the washing up and could party on, but every second contestant behind him had to go to the kitchen (those with numbers 4, 6, 8, etc). Then the next contestant in the remaining queue had to tell his number. He answered 3 and was freed from assisting, but every third contestant behind him was to help (those with numbers 9, 15, 21, etc). The next in the remaining queue had number 5 and was free, but every fifth contestant behind him was selected (those with numbers 19, 35, 49, etc). The next had number 7 and was free, but every seventh behind him had to assist, and so on.

Let us call the number of a contestant who does not need to assist with washing up a lucky number. Continuing the selection scheme, the lucky numbers are the ordered sequence 2, 3, 5, 7, 11, 13, 17, etc. Find out the lucky numbers to be prepared for the next contest party.

Input Specification

The input contains several test cases. Each test case consists of an integer **n**. You may assume that 1<=**n**<=3000. A zero follows the input for the last test case.

Output Specification

For each test case specified by **n** output on a single line the **n-th** lucky number.

Sample Input	Sample Output
1 2 10 20 0	2 3 29 83

Problem B : Electrical Circuit *(orange balloon)*

Physics teachers in high school often think that problems given as text are more demanding than pure computations. After all, the pupils have to read and understand the problem first! So they don't state a problem like "U=10V, I=5A, P=?" but rather like "You have an electrical circuit that contains a battery with a voltage of U=10V and a light-bulb. There's an electrical current of I=5A through the bulb. Which power is generated in the bulb?".

However, half of the pupils just don't pay attention to the text anyway. They just extract from the text what is given: U=10V, I=5A. Then they think: "Which formula do I know? Ah yes, P=U*I. Therefore P=10V*5A=500W. Finished." OK, this doesn't always work, so these pupils are usually not the top scorers in physics tests. But at least this simple algorithm is usually good enough to pass the class. (Sad but true) We will check if a computer can pass a high school physics test. We will concentrate on the P-U-I type problems first. That means, problems in which two of power, voltage and current are given and the third is wanted.

Your job is to write a program that reads such a text problem and solves it according to the simple algorithm given above.

Input Specification

The first line of the input file will contain the number of test cases. Each test case will consist of one line containing exactly two data fields and some additional arbitrary words. A data field will be of the form I=*x*A, U=*x*V or P=*x*W, where *x* is a real number. Directly before the unit (A,V or W) one of the prefixes m (milli), k (kilo) and M (Mega) may also occur. To summarize it: Data fields adhere to the following grammar:

DataField ::= Concept '=' RealNumber [Prefix] Unit
Concept ::= 'P' | 'U' | 'I'
Prefix ::= 'm' | 'k' | 'M'
Unit ::= 'W' | 'V' | 'A'

Additional assertions:

- The equal sign ('=') will never occur in another context than within a data field.

- There is no whitespace (tabs, blanks) inside a data field.
- Either P and U, P and I, or U and I will be given.

Output Specification

For each test case, print three lines:

- a line saying "Problem #*k*" where *k* is the number of the test case
- a line giving the solution (voltage, power or current, dependent on what was given), written without a prefix and with two decimal places as shown in the sample output
- a blank line

Sample Input
3 If the voltage is U=200V and the current is I=4.5A, which power is generated? A light-bulb yields P=100W and the voltage is U=220V. Compute the current, please. bla bla bla lightning strike I=2A bla bla bla P=2.5MW bla bla voltage?

Sample Output
Problem #1 P=900.00W Problem #2 I=0.45A Problem #3 U=1250000.00V

Problem C : Aliens *(yellow balloon)*

The decimal numeral system is composed of ten digits, which we represent as "0123456789" (the digits in a system are written from lowest to highest). Imagine you have discovered an alien numeral system composed of some number of digits, which may or may not be the same as those used in decimal. For example, if the alien numeral system were represented as "oF8", then the numbers one through ten would be (F, 8, Fo, FF, F8, 8o, 8F, 88, Foo, FoF). We would like to be able to work with numbers in arbitrary alien systems. More generally, we want to be able to convert an arbitrary number that's written in one alien system into a second alien system.

Input Specification

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case is a line formatted as : **alien_number source_language target_language**

Each language will be represented by a list of its digits, ordered from lowest to highest value. No digit will be repeated in any representation, all digits in the alien number will be present in the source language, and the first digit of the alien number will not be the lowest valued digit of the source language (in other words, the alien numbers have no leading zeroes). Each digit will either be a number 0-9, an uppercase or lowercase letter, or one of the following symbols **!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~**

Output Specification

For each test case, output one line containing "Alien #x: " followed by the alien number translated from the source language to the target language.

Sample Input	Sample Output
4 9 0123456789 oF8 Foo oF8 0123456789 13 0123456789abcdef 01 AMPHI =HMPAI $NRAI!	Alien #1: Foo Alien #2: 9 Alien #3: 10011 Alien #4: IRAN!

Problem D : Triangle *(black balloon)*

You're interested in writing a program to classify triangles. Triangles can be classified according to their internal angles. If one of the internal angles is exactly 90 degrees, then that triangle is known as a "right" triangle. If one of the internal angles is greater than 90 degrees, that triangle is known as an "obtuse" triangle. Otherwise, all the internal angles are less than 90 degrees and the triangle is known as an "acute" triangle.

Triangles can also be classified according to the relative lengths of their sides. In a "scalene" triangle, all three sides have different lengths. In an "isosceles" triangle, two of the sides are of equal length. (If all three sides have the same length, the triangle is known as an "equilateral" triangle, but you can ignore this case since there will be no equilateral triangles in the input data.)

Your program must determine, for each set of three points, whether or not those points form a triangle. If the three points are not distinct, or the three points are collinear, then those points do not form a valid triangle. (Another way is to calculate the area of the triangle; valid triangles must have non-zero area.) Otherwise, your program will classify the triangle as one of "acute", "obtuse", or "right", and one of "isosceles" or "scalene".

Input Specification

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case is a line formatted as : **x1 y1 x2 y2 x3 y3**

Output Specification

For each test case, output one line containing one of these strings:

- isosceles acute triangle
- isosceles obtuse triangle
- isosceles right triangle
- scalene acute triangle
- scalene obtuse triangle
- scalene right triangle
- not a triangle

Sample Input	Sample Output
8 0 0 0 4 1 2 1 1 1 4 3 2 2 2 2 4 4 3 3 3 3 4 5 3 4 4 4 5 5 6 5 5 5 6 6 5 6 6 6 7 6 8 7 7 7 7 7 7	Triangle #1: isosceles obtuse triangle Triangle #2: scalene acute triangle Triangle #3: isosceles acute triangle Triangle #4: scalene right triangle Triangle #5: scalene obtuse triangle Triangle #6: isosceles right triangle Triangle #7: not a triangle Triangle #8: not a triangle

Problem E : GOLDBACH'S CONJECTURE *(green balloon)*

In 1742, Christian Goldbach, a German amateur mathematician, sent a letter to Leonhard Euler in which he made the following conjecture: Every even number greater than 4 can be written as the sum of two odd prime numbers.

For example:

- 8 = 3 + 5. Both 3 and 5 are odd prime numbers.
- 20 = 3 + 17 = 7 + 13.
- 42 = 5 + 37 = 11 + 31 = 13 + 29 = 19 + 23.

Today it is still unproven whether the conjecture is right.

Anyway, your task is now to verify Goldbach's conjecture for all even numbers less than a million.

Input Specification

The input file will contain one or more test cases. Each test case consists of one even integer n with 6 <= **n** < 1000000. Input will be terminated by a value of 0 for n.

Output Specification

For each test case, print one line of the form **n = a + b**, where **a** and **b** are odd primes. Numbers and operators should be separated by exactly one blank like in the sample output below. If there is more than one pair of odd primes adding up to **n**, choose the pair where the difference **b - a** is maximized. If there is no such pair, print a line saying "Goldbach's conjecture is wrong."

Sample Input	Sample Output
8 20 42 0	8 = 3 + 5 20 = 3 + 17 42 = 5 + 37

Problem F : Member's IDs *(red balloon)*

The CERADEI-GIT (Cercle d'Echange de Réflexion et d'Action Des Elèves Ingénieurs en Génie Informatique et Télécommunications), with a view to record his members' data, wants to create a system of member's IDs allocation. The member's IDs are of the form CCCCVVNNN, where CCCC are the first four consonants of the member's first name, VV are the first two vowels of the member's surname, and NNN is a three-digit sequence number used to distinguish between members who would otherwise have the same member's ID (vowels are the letters A, E, I, O and U, and consonants are the other 21 letters of the English alphabet). If there are not enough consonants in the first name or vowels in the surname, fill the remaining places with Z.

Having the list of CERADEI-GIT members' names, your task will be to determine the matching member's IDs. When assigning the sequence number part of the member's ID, use the smallest number (starting from 000) that will make the member's ID different from all previous.

Example

Suppose the members' names are recorded in the order: ASSOUMA Malik, OUSSOU Julienne, ABO Malik, DADONOUGBO Malaika and SOULE Aziz. The corresponding member's IDs would be MLKZAO000, JLNNOU000, MLKZAO001, MLKZAO002 and ZZZZOU000. ABO Malik has a sequence number of 001, because MLKZAO000 has already been assigned to ASSOUMA Malik. That leaves DADONOUGBO Malaika with MLKZAO002.

Input Specification

The input contains the names of the members, one per line. Each line contains a surname and a first name, separated by a space. Surnames consist only of uppercase English letters and only the first letter of the first name is uppercase, there is no punctuation in names.

The end of input is marked by a line containing only the string `-1'.

The input contains at most 1000 members and the surname and first name of each member is at most 20 letters long.

Output Specification

For each member in the input, output the corresponding member's ID on a separate line. The letters must be in uppercase.

Sample Input	Sample Output
ASSOUMA Malik OUSSOU Julienne ABO Malik DADONOUGBO Malaika SOULE Aziz -1	MLKZAO000 JLNNOU000 MLKZAO001 MLKZAO002 ZZZZOU000

Problem G : Boolean Matrix *(pink balloon)*

A **boolean matrix** has the parity property when each row and each column has an even sum, i.e. contains an even number of bits which are set. Here's a 4 x 4 matrix which has the parity property:

1 0 1 0
0 0 0 0
1 1 1 1
0 1 0 1

The sums of the rows are 2, 0, 4 and 2. The sums of the columns are 2, 2, 2 and 2.

Your job is to write a program that reads in a matrix and checks if it has the parity property. If not, your program should check if the parity property can be established by changing only one bit. If this is not possible either, the matrix should be classified as corrupt.

Input Specification

The input file will contain one or more test cases. The first line of each test case contains one integer **n** (**n**<100), representing the size of the matrix. On the next **n** lines, there will be **n** integers per line. No other integers than 0 and 1 will occur in the matrix. Input will be terminated by a value of 0 for **n**.

Output Specification

For each matrix in the input file, print one line. If the matrix already has the parity property, print "**OK**". If the parity property can be established by changing one bit, print "**Change bit (i,j)**" where **i** is the row and **j** the column of the bit to be changed. Otherwise, print "**Corrupt**".

Sample Input	Sample Output
4 1 0 1 0 0 0 0 0 1 1 1 1 0 1 0 1	OK Change bit (2,3) Corrupt

4 1 0 1 0 0 0 1 0 1 1 1 1 0 1 0 1 4 1 0 1 0 0 1 1 0 1 1 1 1 0 1 0 1 0	

Problem H : Sudoku *(white balloon)*

Sudoku is a popular single player game. The objective is to fill a 9x9 matrix with digits so that each column, each row, and all 9 non-overlapping 3x3 sub-matrices contain all of the digits from 1 through 9. Each 9x9 matrix is partially completed at the start of game play and typically has a unique solution.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Given a completed $\mathbf{N^2}$ x $\mathbf{N^2}$ Sudoku matrix, your task is to determine whether it is a valid solution. A valid solution must satisfy the following criteria:

- Each row contains each number from **1** to $\mathbf{N^2}$, once each.
- Each column contains each number from **1** to $\mathbf{N^2}$, once each.
- Divide the $\mathbf{N^2}$x$\mathbf{N^2}$ matrix into $\mathbf{N^2}$ non-overlapping **N** x **N** sub-matrices. Each sub-matrix contains each number from **1** to $\mathbf{N^2}$, once each.

You don't need to worry about the uniqueness of the problem. Just check if the given matrix is a valid solution.

Input Specification

The first line of the input gives the number of test cases, **T(**$1 \leq \mathbf{T} \leq 100$**)**. **T** test cases follow. Each test case starts with an integer **N (**$3 \leq \mathbf{N} \leq 6$**)**. The next $\mathbf{N^2}$ lines describe a completed Sudoku solution, with each line contains exactly $\mathbf{N^2}$ integers. All input integers are positive and less than 1000.

Output Specification

For each test case, output one line containing "**Sudoku #x: y**", where x is the case number (starting from 1) and y is "Yes" (quotes for clarity only) if it is a valid solution, or "No" (quotes for clarity only) if it is invalid.

Sample Input	Sample Output
3 3 5 3 4 6 7 8 9 1 2 6 7 2 1 9 5 3 4 8 1 9 8 3 4 2 5 6 7 8 5 9 7 6 1 4 2 3 4 2 6 8 5 3 7 9 1 7 1 3 9 2 4 8 5 6 9 6 1 5 3 7 2 8 4 2 8 7 4 1 9 6 3 5 3 4 5 2 8 6 1 7 9 3 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 3 5 3 4 6 7 8 9 1 2 6 7 2 1 9 5 3 4 8 1 9 8 3 4 2 5 6 7 8 5 9 7 6 1 4 2 3 4 2 6 8 999 3 7 9 1 7 1 3 9 2 4 8 5 6 9 6 1 5 3 7 2 8 4 2 8 7 4 1 9 6 3 5 3 4 5 2 8 6 1 7 9	Sudoku #1: Yes Sudoku #2: No Sudoku #3: No