University of Scranton
ACM Student Chapter / Computing Sciences Department
23rd Annual High School Programming Contest (2013)

Problem 1: Footrace

Two runners, A and B (for Ann and Bill, say), are facing each other in a footrace. At certain points in time, measurements are taken to ascertain how far each runner has advanced since the most recent measurement. Each time a measurement is taken, a runner's position corresponds to the sum of that runner's measurements taken up to that moment.

Develop a program that, given a sequence of such measurements, reports which runner is ahead (if either), and by how much, at the time of each measurement. Also, the program is to report each time that a "pass" occurs, which happens when one runner takes the lead in a situation in which the opposite runner was the one leading most recently.

Input: The first line contains a positive integer n indicating how many times measurements were taken for each runner. On each of the next n lines appears a pair of integer measurements, the first of which applies to A and the second to B.

Output: For each pair of measurements, report which runner was ahead, and by how much, when those measurements were taken. If the two runners were even, report that. If a pass occurred, report that. (See sample output for proper formatting.)

Sample input:	Resultant output:
12	
3 5	B ahead by 2
2 3	B ahead by 3
5 2	Even
2 2	Even
4 3	A ahead by 1 (pass)
2 5	B ahead by 2 (pass)
3 2	B ahead by 1
4 3	Even
4 6	B ahead by 2
5 2	A ahead by 1 (pass)
2 4	B ahead by 1 (pass)
4 4	B ahead by 1

Problem 2: Sudoku Address Translation

A **Sudoku** puzzle is played on a 9×9 matrix —with nine embedded 3×3 sub-matrices, or "blocks"— in which each cell either is empty or contains an integer in the range 1..9.

The player's goal is to fill in the empty cells so that each of the integers in the range 1..9 appears exactly once in each row, column, and block.

Our concern here, however, is not with solving a Sudoku puzzle, but rather with translating between two different addressing schemes that can be used for referring to cells in the matrix.

In one scheme, we identify a cell by specifying the row and column in which it lies. In the other, we identify a cell by specifying its block and position (within the block). We call these the **RC** and **BP** addressing schemes, respectively.

In the figure below, a 9×9 matrix is shown on the left. The row numbers are listed vertically to the left of the matrix and the column numbers are listed horizontally above it. The middle cell of each of the nine 3×3 blocks is labeled by the block's number. On the right appears a single 3×3 block in which each cell is labeled by its position within that block.

	0	1	2	3	4	5	6	7	8
0									
1		0			1			2	
2									
3									
4		3			4			5	
5									
6									
7		6			7			8	
8									

0	1	2
3	4	5
6	7	8

The cell in row 3 and column 7, for example, is identified by $RC(3, 7)$. The same cell, identified using its BP-address, is $BP(5, 1)$. Notice that, in both addressing schemes, a cell's address is described by an ordered pair of integers in the range 0..8.

Develop a program that, given a cell's address in either of the two addressing schemes, computes that cell's address as expressed in the other scheme.

Input: The first line contains a positive integer n indicating how many addresses are to be translated. Each of the next n lines contains an address, which is described by either `RC` or `BP` (indicating the addressing scheme) followed by two integers in the range 0..8.

Output: For each address provided as input, a single line of output should be produced showing the given address, followed by an equals sign, followed by the equivalent address in the opposite addressing scheme. (See sample output below for proper formatting.)

```
Sample input:        Resultant output:
-------------        -----------------
5
RC 3 7               RC(3,7) = BP(5,1)
BP 6 4               BP(6,4) = RC(7,1)
RC 2 2               RC(2,2) = BP(0,8)
BP 8 8               BP(8,8) = RC(8,8)
RC 7 0               RC(7,0) = BP(6,3)
```

Problem 3: Time Passages (with apologies to Al Stewart)

Develop a program that, given two clock readings (e.g., 10:45am and 4:17pm) taken less than twenty-four hours from each other, reports how much time passed in going from the first to the second. If the first reading corresponds to a later time of day than the second, assume that the first reading came from one calendar day and the second reading came from the next calendar day.

The answer is to be expressed as a number of hours and minutes, with the former in the range 0..23 and the latter in the range 0..59. As the sample output illustrates, values of 0 and 1 give rise to special cases.

Input: The first line contains a positive integer n and each of the next n lines contains a pair of clock readings, separated by a space. See the sample input below for the format of the clock readings.

Output: For each pair of clock readings, one line of output is to be produced, on which is to be displayed a sentence reporting the two clock readings and the length of time from the first to the second. (See sample output below for proper formatting. Notice how the special cases of 0 and 1 are handled.)

Note: For the purposes of this problem, assume that 12:00am refers to midnight (and thus immediately follows 11:59pm) and that 12:00pm refers to noon (and thus immediately follows 11:59am). According to some "clock experts" (see www.ask.com/wiki/Midnight), this usage is not only ambiguous but also grammatically incorrect. **End of note.**

```
Sample input:              Resultant output:
-------------              -----------------
6
10:45am 4:17pm             From 10:45am to 4:17pm is 5 hours and 32 minutes.
11:25am 7:43am             From 11:25am to 7:43am is 20 hours and 18 minutes.
11:34am 12:00pm            From 11:34am to 12:00pm is 26 minutes.
10:13pm 11:13pm            From 10:13pm to 11:13pm is 1 hour.
11:13pm 12:14am            From 11:13pm to 12:14am is 1 hour and 1 minute.
9:00am 9:00am              From 9:00am to 9:00am is 0 minutes.
```

Problem 4: Line Segment Completion

Develop a program that, given as input the length and slope of a line segment, plus the coordinates of its "left" endpoint P, computes the coordinates of its "right" endpoint Q. (By left endpoint we mean the one having the smaller x-coordinate.)

Recall that the length of the line segment with endpoints (x_1, y_1) and (x_2, y_2) is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

In particular, if $x_2 = x_1 + \Delta x$ and $y_2 = y_1 + \Delta y$, this length is $\sqrt{(\Delta x)^2 + (\Delta y)^2}$.

Recall also that if (x, y) and $(x + \Delta x, y + \Delta y)$ both lie on a line of slope m, then $\Delta y = m \cdot \Delta x$.

Input: The first line contains a positive integer n indicating how many line segments are described thereafter. Each line segment is described on a single line of input containing four real numbers: its length (necessarily a positive number), its slope, and its left endpoint's x- and y-coordinates, respectively.

Output: For each line segment given, three lines of output are to be generated. The first is to identify the line segment's length and slope, respectively. The second is to identify the line segment's left (P) and right (Q) endpoints, respectively. The third is to be blank. See the examples below for the proper labeling and format of the output. Numbers should be accurate to at least four digits after the decimal point, but note that computer arithmetic on real numbers is not exact, so don't be alarmed if your results do not match exactly those given below.

```
Sample input:
-------------
6
1.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0
1.0 -1.0 0.0 0.0
1.41422 -1.0 1.0 1.0
24.5 3.0 -12.0 -2.4
5.843 -0.64 2.3 -4.1
```

```
Resultant output:
-----------------
Length = 1.0  Slope = 0.0
P = (0.0,0.0)  Q = (1.0,0.0)

Length = 1.0  Slope = 1.0
P = (0.0,0.0)  Q = (0.70711,0.70711)

Length = 1.0  Slope = -1.0
P = (0.0,0.0)  Q = (0.70711,-0.70711)

Length = 1.41422  Slope = -1.0
P = (1.0,1.0)  Q = (2.0,0.0)

Length = 24.5  Slope = 3.0
P = (-12.0,-2.4)  Q = (-4.25242,20.84274)

Length = 5.843  Slope = -0.64
P = (2.3,-4.1)  Q = (7.22139,-7.24969)
```

Problem 5: Normal Form Prime Factorization

A *prime* number is any positive integer having *exactly two* distinct divisors among the positive integers: itself and one. (Note that this excludes 1, because it has only itself as a divisor.) The first seven prime numbers are 2, 3, 5, 7, 11, 13, and 17.

Each positive integer can be expressed (in a unique way, according to the Fundamental Theorem of Arithmetic) as a product of powers of the prime numbers. This product is called a *prime factorization*. For example,

$$374556 = 2^2 \cdot 3^1 \cdot 5^0 \cdot 7^4 \cdot 11^0 \cdot 13^1 \cdot 17^0 \cdot 19^0 \cdot 23^0 \cdot 29^0 \cdot \ \ldots$$

Typically, when we write such a product, we omit those factors in which the exponent is zero and we omit any exponent that is 1. Rewriting the above product that way, we get

$$374556 = 2^2 \cdot 3 \cdot 7^4 \cdot 13$$

A prime factorization written in this way is said to be in *normal form*.

Develop a program that, given a positive integer, outputs its prime factorization, in normal form. Due to the limitations of "plain text", there is no nice way of displaying exponents, nor of producing the multiplication operator ($\cdot$). The best we can do is to display a prime factorization on two lines, with the exponents on the first line and the prime factors on the second, lined up properly with the exponents. As for the multiplication operator, we use the asterisk ($*$), as is the case in most programming languages.

For example, the prime factorization above would appear as

```
          2        4
374556 = 2  * 3 * 7  * 13
```

Input: The first line contains a positive integer n indicating how many positive integers are to be factorized into primes. The following n lines contain those integers, one per line.

Output: For each integer given as input, its normal form prime factorization is to be displayed, on two lines, as described above (and as exemplified in the sample output below), followed by a blank line. In particular, each occurrence of an asterisk should be such that exactly one column to its left and one column to its right are blank (in both rows).

Sample input and output appear on the next page.

```
Sample input:
-------------
4
374556
5120
1
18207


Resultant output:
-----------------
          2        4
374556 = 2  * 3 * 7  * 13

        10
5120 = 2   * 5


1 =

         2         2
18207 = 3  * 7 * 17
```

Problem 6: Codeword Messages

Consider the set of strings $C = \{aab, ab, baa, bbaab\}$. This is an example of what is called a *code*; each string in the set is called a *codeword*.

With respect to a code, a *message* is any string that can be formed by taking its codewords and concatenating them, with no restriction upon the order in which the codewords occur or upon how many times each codeword is used.

For example, both *aab* and *baaababbbaababaabbaa* are messages with respect to the code C shown above. That *aab* is a message is obvious, as it is itself a codeword. As for the other string, it can be parsed like this:

$$baa \cdot ab \cdot ab \cdot bbaab \cdot ab \cdot aab \cdot baa$$

Think of the dot ($\cdot$) as being the concatenation operator, and notice that each "factor" in the expression is a codeword in C.

On the other hand, the string *aabbaaababbabb* is *not* a message with respect to C because there is no way to write it as a concatenation of C's codewords.

Develop a program that, given as input a code (i.e., a set of codewords) followed by some "candidate" strings, reports, for each of the latter, whether or not it is a message with respect to the given code.

Input: The first line contains a positive integer m ($m \leq 50$) indicating how many codewords are in the code that is given. Each of the next m lines contains one such codeword, which is a string composed of a's and/or b's. You may assume that the codewords appear in alphabetical order and that no codeword is a proper prefix of any other.

The next line contains a positive integer n indicating how many candidate strings are to be tested for being messages with respect to the given code. Each of the next n lines contains one such string, which is composed of a's and/or b's.

Output: For each of the candidate strings, display it and report whether (`YES`) or not (`NO`) it is a message with respect to the given code. See the sample output for the desired format.

Sample input and output is shown on the next page.

Sample input:	Resultant output:
4	aab : YES
aab	baaababbbaababaabbaa : YES
ab	aabbaaababbabb : NO
baa	aba : NO
bbaab	bbaababababbaaaabbaaaab : YES
5	
aab	
baaababbbaababaabbaa	
aabbaaababbabb	
aba	
bbaababababbaaaabbaaaab	

Problem 7: Codeword Messages, with Proper Prefixes Allowed

Note: This problem is the same as the previous one, with one important exception. Here we lift the restriction that the given code is such that no codeword is the proper prefix of another. **End of note.**

Consider the set of strings $C = \{aa, aab, bab, bb, bbaaa, bbaaaba\}$. This is an example of what is called a *code*; each string in the set is called a *codeword*.

With respect to a code, a *message* is any string that can be formed by taking its codewords and concatenating them, with no restriction upon the order in which the codewords occur or upon how many times each codeword is used.

For example, both *aab* and *babaaaabaabbaaaaaaab* are messages with respect to the code C shown above. That *aab* is a message is obvious, as it is itself a codeword. As for the other string, it can be parsed like this:

$$bab \cdot aa \cdot aab \cdot aa \cdot bbaaa \cdot aa \cdot aab$$

Think of the dot ($\cdot$) as being the concatenation operator, and notice that each "factor" in the expression is a codeword in C.

On the other hand, the string *aabbaaababbabb* is *not* a message with respect to C because there is no way to write it as a concatenation of C's codewords.

Develop a program that, given as input a code (i.e., a set of codewords) followed by some "candidate" strings, reports, for each of the latter, whether or not it is a message with respect to the given code.

Input: The first line contains a positive integer m ($m \leq 50$) indicating how many codewords are in the code that is given. Each of the next m lines contains one such codeword, which is a string composed of a's and/or b's. You may assume that the codewords appear in alphabetical order. You should allow for the possibility that some codewords may be proper prefixes of others.

The next line contains a positive integer n indicating how many candidate strings are to be tested for being messages with respect to the given code. Each of the next n lines contains one such string, which is composed of a's and/or b's.

Output: For each of the candidate strings, display it and report whether (`YES`) or not (`NO`) it is a message with respect to the given code. See the sample output for the desired format.

Sample input and output is shown on the next page.

Sample input:

```
6
aa
aab
bab
bb
bbaaa
bbaaaba
9
aab
aabbbbaaaaabbaaa
bbaaababbbbaabbabaabab
aabbaaababbabb
bba
bbaababababbaaaabbaaaab
bbaaaababbbbbbbbab
bbaaaababbbbbbbbaa
bbaaaababbbbbbbba
```

Resultant output:

```
aab : YES
aabbbbaaaaabbaaa  : YES
bbaaababbbbaabbabaabab : YES
aabbaaababbabb : NO
bba : NO
bbaababababbaaaabbaaaab : NO
bbaaaababbbbbbbbab : YES
bbaaaababbbbbbbbaa : YES
bbaaaababbbbbbbba : NO
```