2012 Consortium for Computing Sciences in Colleges Programming Contest Saturday, November 3rd Southern Polytechnic University Marietta, GA

There are eight (8) problems in this packet. Each team member should have a copy of the problems. These problems are NOT necessarily sorted by difficulty. You may solve them in any order.

Remember input/output for the contest will be from `stdin` to `stdout`. `stderr` will be ignored. Do not refer to or use external files in your source code. Extra white space at the end of lines is ignored, but extra white space at the beginning or within text on a line is not ignored. An extra blank line of output is ignored, but blank lines at the beginning or between lines of text are not ignored.

Have Fun & Good Luck! ☺

Problem 1

Nth Largest Value

For this problem, you will write a program that prints the *Nth* largest value in a fixed sized array of integers. To make things simple, *N* will be 3 and the array will always have 10 decimal integer values.

Input

The first line of input contains a single integer *n*, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each test case consists of a single line containing the test case number, followed by a space, followed by ten decimal integers whose values are between 1 and 1000 inclusive. Each of the ten integers will always be separated by a single space.

Output

For each test case, generate one line of output with the following values: the test case number, a space, and the *3rd* largest value of the corresponding 10 integers.

Sample Input

```
4
1 1 2 3 4 5 6 7 8 9 1000
2 338 304 619 95 343 496 489 116 98 127
3 931 240 986 894 826 640 965 833 136 138
4 940 955 364 188 133 254 501 122 768 408
```

Output Corresponding to Sample Input

```
1 8
2 489
3 931
4 768
```

Problem 2

Fifth Grade Math

In fifth grade, students are presented with different ways to calculate the *Least Common Multiple* (LCM) and the *Greatest Common Factor* (GCF) of two integers. The LCM of two integers *a* and *b* is the smallest positive integer that is a multiple of both *a* and *b*. The GCF of two non-zero integers *a* and *b* is the largest positive integer that divides both *a* and *b* without remainder. For this problem you will write a program that determines both the LCM and GCF for positive integers.

Input

The first line of input contains a single integer *n*, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each test case consists of a single line of input containing two positive integers, *a* and *b*, ($1 \leq a,b \leq 1000$) separated by a space.

Output

For each data set, you should generate one line of output with the following values: the data set number, a space, the LCM, a space, and the GCF.

Sample Input

```
3
5 10
7 23
42 56
```

Output Corresponding to Sample Input

```
1 10 5
2 161 1
3 168 14
```

Problem 3

Breakfast

Chip loves going to Panera for breakfast. He always gets the same thing every morning: a medium cup of coffee and a pastry of some sort. He doesn't always get the same pastry, but he does only pick from a few choices, like a cinnamon roll, or bagel.

To keep his breakfasts a little more exciting, he's decided to never get the same thing twice in a row. So if he gets a cinnamon roll on Tuesday, he won't get one on Wednesday (but he can get one on Thursday). Your job is to calculate how many different ways Chip can have breakfast, given the number of types of pastries he's willing to eat and how many days.

For instance, if Chip wants to only eat a bagel, danish**,** or cinnamon roll over three days he could eat in the following pattern: bagel, danish, roll, and then danish. However, he doesn't have to eat all of the pastries, so he could eat like this: roll, danish, roll, and then danish. Chip only requires that he doesn't eat the same thing twice in a row.

Input

The first line of input will contain an integer n telling how many cases will follow. The next n lines will contain an integer p, ($1 \leq p \leq 100$), and another integer d, ($1 \leq d \leq 7$). p is how many types of pastries that Chip will eat, and d is the number of days to calculate. p and d are separated by a single space.

Output

For each line of input, you should output a value k, which is the number of different ways Chip can have breakfast for that input.

Sample Input

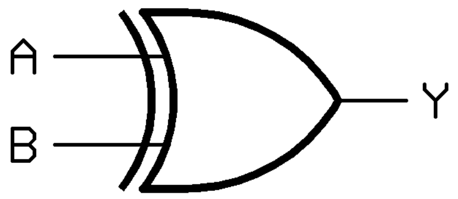
```
3
2 3
3 2
3 4
```

Output Corresponding to Sample Input

```
2
6
24
```

Problem 4

XOR Reduction



You've been designing a digital logic circuit as of late and want to prune it down some to make it more manageable. Your circuit consists only of XOR gates with two inputs each, and some static inputs of true and false. Recall that the XOR outputs true if and only if one of the values is true and not the other. If both values are true or both values are false, the XOR gate outputs false.

Input

The first line of input contains a single integer *n*, ($0 \leq n \leq 1000$), which is the number of gate(s) in the circuit. This is followed by *n* line(s) each with three values on it separated by a single space. The first value will be the index *k* of the gate ($0 \leq k \leq n\text{-}1$). The second value will be either an index of a gate whose output connects to it or a `T` or `F` value indicating a static input. The third value will be in the same format as the second: an index for a gate or a `T/F` value (the second connected value to the gate).

This is followed by a line that starts with a gate index. For this gate, you should calculate the output. You are guaranteed there will be no cycles in the circuit and that the last index provided can be calculated.

The last input case will have $n = 0$. Do not process this case.

Output

You should print out either `T` or `F` for whether the gate outputs true or false.

Sample Input

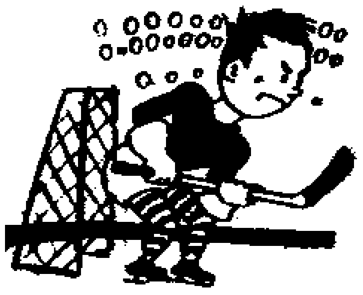
```
1
0 T T
0
2
0 F T
1 0 F
1
0
```

Output Corresponding to Sample Input

```
F
T
```

Problem 5

Winning Goal



In team sports, every player has an essential role to play. But when looking back on an exciting game, one of the greatest highlights is the goal that clinched the victory. In this problem, you will write a program that reads a list of goals scored during several different hockey games. For each game, you are to determine its final score and the name of the player responsible for scoring the winning goal.

We define the winning goal of a game as follows. If the winning team scored *W* goals, and the losing team scored *L* goals, then the winning goal is the $(L+1)^{st}$ goal scored by the winning team. For example, if Los Angeles beat Phoenix by a score of 6 to 3, then LA's 4th goal is the winning goal.

Also note that the order in which the goals are scored does not matter. Phoenix could have scored all of its goals before LA began scoring, or vice versa. The winning goal is always the $(L+1)^{st}$ goal scored by the winning team.

You may assume that there was at least one goal scored in each game, and that no game ended in a tie.

Input

The first line of input is of the form :

```
 n games
```

and indicates the number of games. There will always be at least two games, but no more than 20. Each game is introduced by a line of the form:

```
Game <n> <team> vs <team>
```

where `<n>` is a positive integer, and `<team>` is a 3-letter abbreviation. Do not assume that the order in which the teams are identified has any bearing on which team won the game. The remaining lines of a game identify goal events. Each such line has this format:

```
<2 spaces> <team> (<player number>) <player name>
```

These lines will be indented by two spaces, and indicate the name (abbreviation) of the team that scored the goal, followed by the number and name of the scoring player. The player number is given inside parentheses. Note that the player's name may contain several words, but does not extend past the end of the line.

The end of input is signified by an asterisk given at the beginning of a line.

Output

For each game, your program needs to print a line of information, giving the game number, the score, and the name of the player who scored the winning goal. The format must be:

```
Game <n>, <win team> <win score> <lose team> <lose score>, winning goal by <player>
```

Note that your program must determine which team won the game, and output the winning team's abbreviation and score before those of the losing team. At the end of the line, print the player's name but not the player's number. Also, note the commas required in the output format.

Sample Input

```
2 games
Game 1 TOR vs TBL
  TOR (14) Matt Stajan
  TBL (16) Dixon Ward
  TBL (4) Vincent Lecavalier
Game 2 OTT vs MTL
  MTL (46) Andre Kostitsyn
  MTL (21) Chris Higgins
  MTL (46) Andre Kostitsyn
  OTT (20) Antoine Vermette
  MTL (51) Francis Bouillon
  MTL (79) Andrei Markov
  MTL (54) Mikhail Grabovski
  MTL (6) Tom Kostopoulos
  OTT (15) Dany Heatley
  OTT (28) Martin Lapointe
  OTT (15) Dany Heatley
  OTT (19) Jason Spezza
*
```

Output Corresponding to Sample Input

```
Game 1, TBL 2 TOR 1, winning goal by Vincent Lecavalier
Game 2, MTL 7 OTT 5, winning goal by Mikhail Grabovski
```

Problem 6

Conversions

Conversion between the *metric* and *English* measurement systems is relatively simple. Often, it involves either multiplying or dividing by a constant. You must write a program that converts between the following units:

Type	Metric	English equivalent
Weight	1.000 kilograms	2.2046 pounds
	0.4536 kilograms	1.0000 pound
Volume	1.0000 liter	0.2642 gallons
	3.7854 liters	1.0000 gallon

Input
The first line of input contains a single integer *n*, (1 ≤ *n* ≤ 1000), which is the number of test cases that follow. Each test case consists of a single line of input containing a floating point (double precision) number, a space and the *unit specification* for the measurement to be converted. The *unit specification* is one of `kg`, `lb`, `l`, or `g` referring to kilograms, pounds, liters and gallons respectively.

Output
For each test case, you should generate one line of output with the following values: the test case number as a decimal integer (start counting at one), a space, and the appropriately converted value rounded to 4 decimal places, a space and the *unit specification* for the converted value.

Note that the common method for rounding should be used. If the digit in the place you are rounding is followed by 0, 1, 2, 3, or 4, round *down*. If it is followed by 5, 6, 7, 8, or 9, then round *up*.

Sample Input
```
5
1 kg
2 l
7 lb
3.5 g
0 l
```

Output Corresponding to Sample Input
```
1 2.2046 lb
2 0.5284 g
3 3.1752 kg
4 13.2489 l
5 0.0000 g
```

Problem 7

Tweening

You've been using jQuery for all your web development needs lately, and you were saddened to discover that despite all of jQuery's bells and whistles, jQuery doesn't allow color animation. For many CSS properties, like opacity, height, width, etc., you can specify a start and stop point for the animation and a time it takes to do the animation and jQuery will handle the rest. With colors, this just isn't the case.

Being the resolute programmer that you are, you've decided to write your own jQuery extension to do color in-betweening. You've been reading the API, and you've discovered that jQuery uses linear interpolation. So given two hex colors and a duration, you need to take each color component separately and interpolate linearly. Then, you'll need to combine them back again into an HTML color of the same format as the originals. However, the result of interpolation might not be an integer, so you have decided to always truncate to an integer.

Consider the case with `#ff0000` (red) and `#00ff00` (green) with a duration of 5000 milliseconds in length, and the middle time to interpolate for was 2500 milliseconds, the color would be `#808000`. This is because you are halfway between the start and stop time, so the red component will be halfway between `ff` (255) and `00` (0) which is 127.5 (which we truncate to 127, giving `7f` in hex), and the green will be halfway between `00` and `ff` giving the same result. The blue remains 0 because both endpoints are 0.

Input
Input will consist of a single line with an integer *n*, telling how many cases will follow. The next *n* lines will each have four tokens separated by a single space : a start hex color, a stop hex color, a duration time, and a middle time to calculate for. The duration time is guaranteed to be positive, and the middle time between 0 and the duration time.

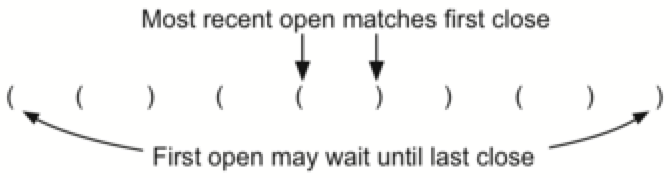
Output
For your output, you should print the interpolated hex color on a single line. If during interpolation, the values become fractional, you should round to the nearest integer.

Sample Input
```
3
#ff0000 #00ff00 5000 2500
#ffffff #000000 100 30
#ff6666 #6666ff 1000 800
```

Output Corresponding to Sample Input
```
#7f7f00
#b2b2b2
#8466e0
```

Problem 8

Balanced


Most recent open matches first close
(( ) (( )) ( ))
First open may wait until last close

We want to write a program that takes a string of opening and closing parentheses and checks to see whether it's *balanced*. We have exactly two types of parentheses for this problem: round brackets: () and square brackets: []. Assume that the string doesn't contain any other characters than these. This means no spaces, digits, letters, or other symbols.

Balanced parentheses require that there are an equal number of opening and closing parentheses. It requires that every opening parenthesis be closed in the reverse order opened. For example, `([])` is balanced, but `([)]` is not. It also requires that no closing parenthesis appears before an opening parenthesis.

Input
The file contains a positive integer *n* and a sequence of *n* strings of parentheses, one string per line. Each string is of length *n*, ($2 \leq n \leq 80$). You may assume each string contains only parentheses of type `()` and `[]`, and no other characters.

Output
Output each input string in the format below indicating whether or not it is balanced.

Sample Input
```
8
()
[]
([])
([()[]()])()
(([()])))
)(
(()(())()))
([)]
```

Output Corresponding to Sample Input
```
() is balanced
[] is balanced
([]) is balanced
([()[]()])() is balanced
(([()]))) is not balanced
)( is not balanced
(()(())()) is balanced
([)] is not balanced
```