Consortium for Computing Sciences in Colleges
2023 Southeastern Programming Competition
Saturday, November 4th, 10 AM – 1 PM EDT

There are nine (9) problems in this packet. 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. Here are some reminders for our competition.

- Each team may comprise 2–4 students, but only up to two students can be at a machine at a time.
- Internet access is only allowed to consult the APIs of a programming language. Cell phone usage is not allowed at any time.
- Any team found to be communicating with someone other than their teammates for assistance will be disqualified.
- Four programming languages are allowed: C#, C++, Java, Python
- An awards ceremony will be held at 1:30 PM EST.
- Have a lot of fun and good luck! 😊

Problem 1

The Myrtle Beach Marathon

The Myrtle Beach Marathon is an event held in South Carolina every year. Every first Saturday in March, between 5,500 and 7,500 people come together to compete in the 26.2 mile race. It stands as the biggest marathon in South Carolina, utilizing a flat, speedy, and enjoyable course that stretches along the picturesque Grand Strand.

In order to prepare for the marathon, you plan to do some training on the firm sand of the beaches of Cocoa Beach, Florida, as well as outside the United States along the shores of Cancun, Mexico. In Mexico, mileage is marked in kilometers.

You will write a program that will let one enter the distances and times of your training runs on different days, and will calculate and print out a summary of the miles, kilometers, time, and pace per mile and kilometer of each day. In addition, the overall distance and time run will be shown. Note that there are 1,609.34 meters in one mile and 1,000 meters in one kilometer.

Input

The input to your program will contain at least one day of training distances and times. The last line in each day of running is the line `LAST` and should not be processed. The end of all input is indicated by the line `END` and should not be processed. All lines of data from a particular day of training consist of one or more distances followed by a single lowercase letter "`m`" or "`k`". The letter "`m`" indicates miles, and the letter "`k`" indicates kilometers. This is followed by a single space, and then the total time in hours, minutes, and seconds (`hh:mm:ss`).

Output

The program should print a summary in tabular format as shown on the next page. Include headings on the first line for the miles, kilometers, time, pace/mile, and pace/km of each leg. This is followed by a blank line and then the label `Day`, followed by the day number and corresponding statistics for that particular day. All pace output should be in the format `hh:mm:ss`. Lastly, print a blank line followed by a line with the overall distance for all days in miles and kilometers and a line with the overall time spent running in days, hours, minutes, and seconds (`dd:hh:mm:ss`). You may assume the total time will be less than 100 days.
All data output should be left-justified. Output distances to two decimal points. All data output should be lined up in columns. Use one tab between each column, except for `Pace/mile` and `Pace/km` which are each preceded by two tabs. Your output should not be more than 80 columns wide. Column headers should be aligned above the data as shown.

Sample Input

```
13.1m 02:24:44
13.1m 02:24:59
LAST
26.2m 04:29:53
LAST
8k 00:41:41
8k 00:39:53
8k 00:41:00
5k 00:24:14
5k 00:23:29
5k 00:23:25
LAST
42k 03:41:44
LAST
20m 02:34:11
LAST
END
```

Output Corresponding to Sample Input

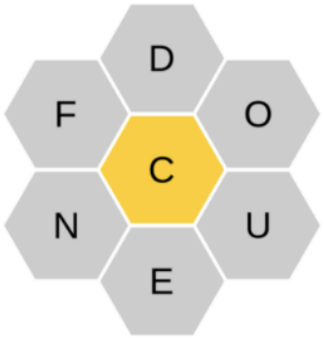
```
        Miles   Km      Time            Pace/mile       Pace/km

Day1    26.20   42.16   04:49:43        00:11:03        00:06:52
Day2    26.20   42.16   04:29:53        00:10:18        00:06:24
Day3    24.23   39.00   03:13:42        00:07:59        00:04:58
Day4    26.10   42.00   03:41:44        00:08:29        00:05:16
Day5    20.00   32.19   02:34:11        00:07:42        00:04:47

Overall Distance of 122.73 miles (197.52 km)
Overall Time of 00:18:49:13
```

Problem 2

The Spelling Bee



Spelling bee is a game with a 7-letter hexagon at the center. Players get points by forming words with four or more letters that include the letter in the center of the grid. A single letter can be used multiple times and each puzzle will have at least one *pangram* – a word that uses all 7 letters. For example, `COFOUNDED` is a pangram using the hexagon here. Four-letter words will get you one point. Starting with five-letter words, players receive one point per letter. Pangrams earn seven additional points. For example, `CEDE` is 1 point, `CODED` is 5 points, and `COFOUNDED` is 16 points. The game offers a hint that displays the number of words that start with each letter, along with how many words there are of each length. This information is organized in the form of a frequency chart as seen below. You are also told the number of words, maximum score, and number of pangrams.

	4	5	6	7	8	9	10	*tot*
C	6	6	4	3	4	1	1	***25***
D	1	2	2	2	1	1	-	***9***
E	-	1	2	1	-	-	-	***4***
F	-	1	2	-	-	-	-	***3***
N	-	1	1	-	-	-	-	***2***
O	1	1	-	-	-	-	-	***2***
U	-	-	1	1	1	-	-	***3***
tot	***8***	***12***	***12***	***7***	***6***	***2***	***1***	***48***

words: 48
score: 279 pts
pangrams: 2

Your job is take an input file of all possible words formed by a given spelling bee and print a frequency chart like the one above along with all the pangrams and the maximum score.

Input

The first line of input will contain a single integer n, which represents the number of words in the input file. You may assume that $1 \leq n \leq 100$. This will be followed by n words in sorted order each on a separate line. For each word, the first letter will always be capitalized. All other characters will be lowercase and alphabetic. All words will be of length l where $4 \leq l \leq 12$.

Output

Your program should output a nine row by eleven column frequency matrix in the format below. The first row acts as a header for each of your lengths l starting at 4 and ending at 12 along with the total sum of frequencies on that row. Rows two through eight represent the frequencies of each of your seven letters. The final row represents sums of each of your ten columns. This is followed by a blank line, each word that is a pangram on separate lines in sorted order, and the maximum score. All frequencies and column headers less than ten should be followed by two spaces; all others are followed by one space. Each letter at the start of rows two through eight is followed by three spaces, while the word `Sum` on line nine is followed by one space.

Sample Input

```
48
Cede
Ceded
Cocoon
Cocooned
Code
Coded
Codon
Coed
Coffee
Cofound
Cofounded
Concede
Conceded
Condo
Condone
Condoned
Cone
Coned
Confound
Confounded
Conned
Cooed
Cued
Cuff
Cuffed
Deco
Decode
Decoded
Deduce
Deduced
Denounce
Denounced
Deuce
Dunce
Educe
Educed
Encode
Encoded
Fecund
Fence
Fenced
Neocon
Nonce
Once
Ounce
Unceded
Uncuff
Uncuffed
```

Output Corresponding to Sample Input

```
    4  5  6  7  8  9  10 11 12 Sum
C   6  6  4  3  4  1  1  0  0  25
D   1  2  2  2  1  1  0  0  0  9
E   0  1  2  1  0  0  0  0  0  4
F   0  1  2  0  0  0  0  0  0  3
N   0  1  1  0  0  0  0  0  0  2
O   1  1  0  0  0  0  0  0  0  2
U   0  0  1  1  1  0  0  0  0  3
Sum 8  12 12 7  6  2  1  0  0  48

Cofounded
Confounded
279 pts
```

Problem 3

Time to Accelerate!

Jessica is majoring in mechanical engineering. In her freshman year, she is currently taking a physics class. One type of problem that Jessica is most interested in is calculating how long it takes for a car to accelerate within a certain distance. For example, if we want to speed up from 40 m.p.h. to 60 m.p.h, and we want to accomplish this in a distance of 0.3 miles, how long would that take? The answer, when rounded to the nearest second, is 22 seconds. Throughout this problem, we will assume uniform acceleration, otherwise known as constant acceleration. You will write a program that will help Jessica verify her answers to problems involving this kind of uniform acceleration.

Input

The first line of the input will contain a positive integer $n \leq 100$, representing the number of test cases. Each of the remaining n lines of input represents a test case. Each test case consists of three real numbers:

- v_1, the initial speed before the acceleration
- v_2, the final speed at the end of the acceleration
- d, the distance in miles over which the acceleration took place

You may assume that $v_2 > v_1$.

Output

For each test case, your program needs to output the amount of time it takes to perform this acceleration. Your output must be rounded to the nearest second, and be shown in the format mm:ss. In other words, the number of minutes and seconds should be separated by a colon with no intervening spaces. The number of seconds must be printed as a two digit number. If the number of seconds is less than ten, then a leading zero is required. If the time is less than 1 minute, than your output must display the number of minutes as zero.

Sample Input

```
3
40 60 0.3
45 65 1
46.5 52.25 0.5
```

Output Corresponding to Sample Input

```
0:22
1:05
0:36
```

Problem 4

Adjective Order

Many times in English prose, we use two adjectives to modify a noun. For example, "big red apple." Or should it be "red big apple?" It turns out that "big red apple" is correct idiomatic English, while "red big apple" is not. If we want the adjectives "big" and "red" to modify "apple," then "big" must come before "red."

According to the Cambridge English Dictionary, the rules determining which of two adjectives should appear first can be established as follows. Adjectives are classified into the following ten categories:

Category	Description	Four example adjectives of each category
1	opinion	`beautiful, delicious, reliable, unusual`
2	size	`big, deep, small, tall`
3	physical quality	`rough, shiny, thin, untidy`
4	shape	`oval, round, square, rectangular`
5	age	`elderly, old, young, youthful`
6	color	`blue, green, red, yellow`
7	origin	`American, Chilean, Dutch, Japanese`
8	material	`brass, plastic, steel, wooden`
9	type	`four-sided, general-purpose, L-shaped, U-shaped`
10	purpose	`cleaning, cooking, frying, hammering`

When multiple adjectives modify a noun, each adjective must come from a higher-numbered (or equal) category than the adjective that precedes it. In other words, suppose a phrase has this structure: $\text{Adj}_1\ \text{Adj}_2$ Noun, showing two adjectives before a noun. Let $C(\text{Adj}_i)$ be the category number $(1–10)$ of adjective number i. Then, we must verify that $C(\text{Adj}_1) \leq C(\text{Adj}_2)$. If $C(\text{Adj}_1) > C(\text{Adj}_2)$, then the adjective order is incorrect.

For example, in the phrase `big red apple`, `big` comes from category 2, and `red` comes from category 6. Since $2 \leq 6$, `big` and `red` are in the correct order when we say `big red apple`.

You need to write a program that will check the appropriate order of adjectives in cases where a noun is being modified by two adjectives.

Input

The first line of the input will contain a single positive integer *n*, which less than 100. This represents the number of test cases. Each of the next *n* lines of input contains one test case. A test case is a three-word phrase of two adjectives followed by a noun. Words may be hyphenated. The only adjectives that you will ever see in the input are listed in the third column of the table above. You may also assume that an adjective belongs to a single category.

Output

For output, your program needs to indicate whether each phrase has its adjectives in the correct order or not. Suppose the format of an input test case is `A1 A2 N`. If the adjective order is correct, then the output format should be:

```
"A1 A2 N" is correct
```

But if the adjective order is incorrect, the output format should be:

```
"A1 A2 N" is incorrect – should be "A2 A1 N"
```

Note that the quotation marks are required in the output.

Sample Input

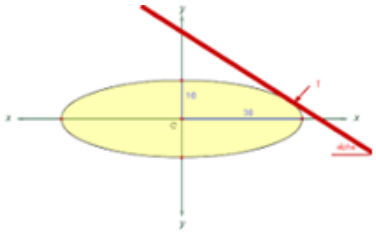
```
3
big tall statue
shiny big car
rectangular cleaning brush
```

Output Corresponding to Sample Input

```
"big tall statue" is correct
"shiny big car" is incorrect - should be "big shiny car"
"rectangular cleaning brush" is correct
```

Problem 5

Tangent Line to Ellipse



One question that frequently arises in calculus is finding the equation of a line that is tangent to an ellipse at a particular point. You will write a program that solves some problems of this type. Let us assume that an ellipse is centered at the origin $(0,0)$. Then, the general equation of the ellipse is:

$$Ax^2 + By^2 = C$$

Assuming that the point (d, e) lies on the ellipse, it is possible to find the slope of the line tangent to the ellipse at this point using standard techniques of differentiation, and from there we can write the equation of the tangent line. For example, the equation of the line tangent to $9x^2 + 12y^2 = 129$ at the point (-3, 2) is $y = 1.125x + 5.375$.

Input

The first line of the input will contain a positive integer *n*, which will be less than 100. Each of the next *n* lines of input will contain five real numbers, A, B, C, d, e, as described above. These five numbers will be separated from each other by one or more spaces. You may assume A through C are positive and (d, e) lies on the ellipse. Your program needs to find the equation of the tangent line, and display it in slope-intercept form: $y = mx + b$. You may assume that the tangent line is neither horizontal nor vertical. Note that when creating test cases, a correct value of C is necessary in order to ensure that the given point (d, e) lies on the ellipse. In solving this problem, you will find that C is never used.

Output

Each test case will have one line of output. A line of output should be formatted as follows.

```
Line #k:  y = mx + b
```

where *k* indicates the number of the test case, 1 through *n*. The values of *m* and *b* are to be rounded to the nearest thousandth, and displayed to exactly three decimal places. Note that if the y-intercept is negative, the plus sign must be replaced with a minus sign to indicate a negative number. Print two spaces after the colon, one space on each side of the equals sign, and one space on either side of the plus or minus sign.

Sample Input

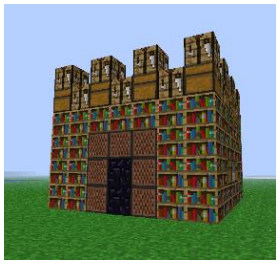
```
2
9   12   129     -3     2
1.5  2.5   0.685  0.2  -0.5
```

Output Corresponding to Sample Input

```
Line #1:  y = 1.125x + 5.375
Line #2:  y = 0.240x - 0.548
```

Problem 6

Stacks of Material



Steve is trying to figure out how much raw material he needs to complete the builds he is working on. Simple builds were easy to calculate. For example, Steve built a house that required 64 wooden planks. Since a wooden log produces four planks, he calculated that he needed to gather sixteen logs. Things got more complex when he started using a variety of different materials like stairs and slabs. A total of six planks are required to craft four stairs, and three planks are used to craft six slabs.

So, when a house remodel required an additional 64 planks, 16 stairs, and 12 slabs; he needed 64 + 24 + 6 = 94 planks in total. That requires 24 logs (of course, we needed to round up, to make sure we have enough). Steve is tired of doing all of these calculations himself and wants you to help him. He will tell you the types of raw material he wants to gather, the recipes to convert materials, and how much material he needs for his builds. He wants you to tell him how much raw material he needs to gather. Note that there will always be *only one way* to craft a material. Every material will either be considered raw, or there will be exactly one recipe that outputs it.

Input

The first line of input will be positive integer $n \leq 100$ denoting the number of cases to consider. The first line of each case will be the name of the build. The following line will be a set of positive three integers, $m, r, p \leq 100$. The next m lines will be a list of raw materials. Each of these lines corresponds to a single material that is considered raw. The following r lines will be a list of recipes. Each recipe line will contain the following: a number of comma separated inputs to the recipe, a greater than sign, and the output to the recipe. Each input and output is preceded by the number of materials required or generated. Finally, the following p lines are a list of products required in the build. Each of these lines will be a material preceded by the number of that material required.

Output

For each case, you should output the name of the build and all of the required raw materials. The materials should be listed next in alphabetical order on separate lines. Each of these lines should start with two spaces, followed by the number of that material required, a single space, then the name of that material. Keep in mind that the number of materials should always be a positive integer that represents the minimum required. Do not print any raw materials that are unnecessary to the build.

Sample Input

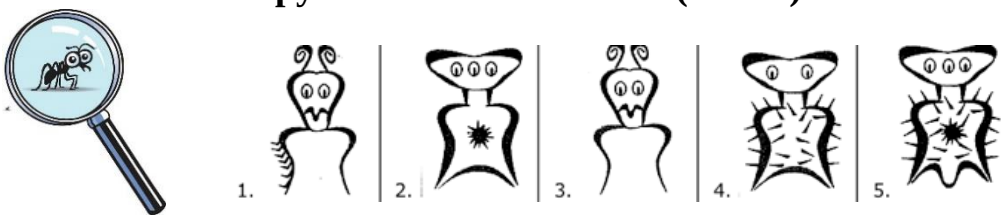
```
2
Wooden Stairs and Slabs
2 3 3
cobblestone
log
1 log > 4 plank
3 plank > 6 slab
6 plank > 4 stair
64 plank
16 slab
12 stair
A Stack of Pistons
4 3 1
oak-log
cobblestone
iron-nugget
redstone-dust
1 oak-log > 4 oak-plank
3 oak-plank, 4 cobblestone, 1 iron-ingot, 1 redstone-dust > 1 piston
9 iron-nugget > 1 iron-ingot
64 piston
```

Output Corresponding to Sample Input

```
Wooden Stairs and Slabs
  23 log
A Stack of Pistons
  256 cobblestone
  576 iron-nugget
  48 oak-log
  64 redstone-dust
```

Problem 7

Creepy Critters Science Club (CCSC)



The Creepy Critters Science Club (CCSC) is very popular with kids, and there are multiple chapters around the country. This year the annual CCSC convention was held at Leech Lake in beautiful Mosquito Acres, Minnesota. Each child was given a magnifying glass and five specimen jars as souvenirs. The magnifying glasses and specimen jars are identical.
But, you know how kids are. When it is time to go home, they just throw things in their bags. And when they got home, they realized that some had forgotten some or all their souvenirs, and some had taken souvenirs that were not theirs.

This would not be a problem if items could be properly distributed within each of the chapters. For example, if there are three children in a chapter, and those children brought home a total of three magnifying glasses and fifteen specimen jars, they can be correctly distributed at the next local chapter meeting.

Your task is to determine whether items can be properly distributed within all chapters.

Input
The first line contains an integer c ($2 \leq c \leq 100$), the number of test cases.
For each test case, there follows:

- A line with an integer n ($2 \leq n \leq 10{,}000$), the number of children at the Leech Lake convention. (children are identified by the integers $0..n-1$)
- n lines (one for each child $i, 0 \leq i \leq n-1$) each containing two integers m and s, the number of magnifying glasses and specimen jars that child i brought home.
- A line with an integer p ($0 \leq p \leq 10{,}000$), indicating the number of u, v pairs to follow. Each u, v pair indicates that child u and child v are in the same chapter.
- p lines, each containing u and v ($0 \leq u, v \leq n-1$)

Output
For each test case, print the case number in the format below followed by a single colon, a single space, and then print `yes` if the items can all be properly distributed within a chapter, otherwise print `no` as follows.

```
Case #c: yes/no
```

Sample Input

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

Output Corresponding to Sample Input

```
Case #1: yes
Case #2: no
```

Explanation of Sample Input and Corresponding Output

The first line indicates there are two test cases. For the first test case:

- There are 5 children.
- Child 0 brought home 0 magnifying glasses and 7 specimen jars. Child 1 brought home 2 magnifying glasses and 3 specimen jars. Child 2 brought home 1 magnifying glass and 5 specimen jars. Child 3 brought home 2 magnifying glasses and 4 specimen jars. Child 4 brought home 0 magnifying glasses and 6 specimen jars.
- There are 3 pairings. Children 0 and 1 are in the same chapter. Children 1 and 2 are in the same chapter. Children 3 and 4 are in the same chapter.
- For the chapter containing children 0, 1, and 2, there is a total of 3 magnifying glasses and 15 specimen jars. For the chapter containing children 3 and 4, there is a total of 2 magnifying glasses and 10 specimen jars. So for all chapters, items can be distributed at the next chapter meeting. So the answer is "yes."

For the second test case:

- There are 3 children.
- Child 0 brought home 0 magnifying glasses and 0 specimen jars. Child 1 brought home 1 magnifying glass and 4 specimen jars. Child 2 brought home 1 magnifying glass and 5 specimen jars.
- There is 1 pairing. Children 0 and 1 are in the same chapter. Child 2 is the only member of a chapter.
- For the chapter containing 1 child, the correct number of magnifying glasses and specimen jars have been brought home. For the chapter containing 2 children, a total of 1 magnifying glass and 4 specimen jars have been brought home, which cannot be distributed correctly. So the answer is "no."

Problem 8

Olympic Dilemma

Mr. Olympia is training for his next meet and needs some help with training. When he is working out, he typically has a target weight he wants to load on his barbell. However, he does not have many plates so sometimes he cannot make that weight exactly. So, he will typically load the barbell to be as close to that weight as possible without going over to prevent injury.

As you can imagine, as he is bought more plates, figuring out how to load the barbell has become challenging. So, he has asked you to write a program to figure out how to load his barbell. In particular, he will give you a target weight and a set of plates. From there, he wants you to provide the *best* way to load the barbell. That is,

- It should be as close as possible to the target weight without going over.
- If there are two possible ways to load barbell, pick one with the least number of plates.
- Be greedy. Load the largest available weights first.

Note that the barbell already weighs 45 pounds. So, if you are targeting 100 pounds, you only need to account for 55 pounds of plates. You may assume that all plates need to be added in pairs. So, if your weight set has three ten pound weights, you really can only use two of them since the barbell has to be balanced on both sides equally.

Input

Input will start with a positive integer *N* denoting how many cases are to follow. A case is structured as follows: It starts with a positive integer *P* denoting how many types of plates Mr. Olympia has. The next *P* lines will contain two positive integers representing the weight *W* and the number *C* of plates of that type, each separated by a single space. After the list of plates, there will be a number of scenarios for that set of plates.

The next line will be a positive integer *S* denoting the number of scenarios, followed by *S* lines. Each line will have a single positive integer denoting a target weight. Mr. Olympia works out a lot, but he does not do many lifts. There will be at most ten total cases, and each case will never have more than 40 total plates, and five scenarios.

Output

For each scenario, output the plate type and number of plates of that type which Mr. Olympia should use. Note that if the target weight is below 45, there is no way to load the barbell and stay under that weight. In that case, output "`No solution`." Also, in some cases, Mr. Olympia will not need to add any plates to hit his target weight. In that case, output "`No plates`." Print all weights in *descending order* in the format `plate_type x number_of_plates`. Use a comma followed by a single space between each weight as shown below.

Sample Input

```
1
3
10 4
20 2
25 2
5
10
50
100
150
200
```

Output Corresponding to Sample Input

```
No solution
No plates
25 x 2
25 x 2, 20 x 2
25 x 2, 20 x 2, 10 x 4
```

Problem 9

The Munchkins

The Munchkin people of Munchkin Country in the Land of Oz are a fun, happy, peace-loving people who have a big problem. They have very little depth perception and are constantly running into doors. They have asked you to write a program to let them know when they should duck going through a doorway. Since Munchkins love fun, some of their doorways have been made intentionally short to force other Munchkins to either crawl or limbo through the door. Unfortunately, the Munchkins do not all use the same set of measurements, so some heights are measured in inches, others in feet, others in yards, others in centimeters, and others in meters. To do the conversions, recall that 1 inch equals 2.54 centimeters, 1 foot equals 12 inches, 1 yard equals 3 feet, and 1 meter equals 100 centimeters.

Input

The input will begin with a single line, telling you how many Munchkins want help. Then, for each Munchkin, there will be data about the Munchkin and the doorways they reach. The first line of each dataset will give you the Munchkin's name (all Munchkins have names of exactly six characters), an integer *n*, the number of doors to process for this Munchkin ($n > 0$), and the Munchkin's height, a floating point number followed by exactly one blank space and then one of `i`, `f`, `y`, `c`, or `m` (representing inches, feet, yards, centimeters, and meters). The next *n* lines will have the height of the doorways in the bumpkin's life, one per line. Each height will be a floating point number followed by exactly one blank and one of `i`, `f`, `y`, `c`, or `m` (representing inches, feet, yards, centimeters, and meters).

Output Corresponding to Sample Input

For each Munchkin, print the name of the Munchkin on one line and then for each door, print the way the Munchkin should travel through the door. Decide on a mode of travel based on the table below, where *b* is the height of the Munchkin, *d* is the height of the door (expressed in the same units) as shown on the table below.

Door height	Travel method
$d > b * 1.25$	`Stilts`
$b * 1.25 >= d > b * 1.05$	`Walk`
$b * 1.05 >= d > b * 0.65$	`Duck`
$b * 0.65 >= d > b * 0.40$	`Crawl`
$b * 0.40 >= d > b * 0.25$	`Limbo`
$b * 0.25 >= d$	`Blocked`

Have one blank line after each data set.

Sample Input

```
2
Mookin 3 150.4 c
75 i
2 f
151 c
Kimkin 1 67.3 i
204.5 c
```

Sample Output

```
Mookin
Doorway 1: Stilts
Doorway 2: Crawl
Doorway 3: Duck

Kimkin
Doorway 1: Walk
```