CATEGORIZED UVA PROBLEMS

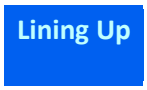
Problem Type : Geometry
Set : GEO-01
Source : UHUNT (Basic Geometry-1st Edition)

Xplosive
Email: nasif.002@gmail.com

Geometry Basics (1/21 = 4%)			
Lines (0/3)			
270 - Lining Up	★	Lev 3	--- ? ---
10263 - Railway	★	Lev 4	--- ? ---
11227 - The silver bullet.	★	Lev 4	--- ? ---
Circles (only) (0/3)			
10005 - Packing polygons	★	Lev 4	--- ? ---
10012 - How Big Is It?	★	Lev 3	--- ? ---
10451 - Ancient Village Sports	★	Lev 3	--- ? ---
Triangles (plus Circles) (0/3)			
190 - Circle Through Three Points	★	Lev 2	--- ? ---
10286 - Trouble with a Pentagon	★	Lev 3	--- ? ---
11152 - Colourful Flowers	★	Lev 3	--- ? ---
Rectangles (1/3)			
476 - Points in Figures: Rectangles	★	Lev 2	--- ? ---
11207 - The easiest way	★	Lev 4	--- ? ---
11455 - Behold my quadrangle	★	Lev 3	✔ 0.009s/1531
Great Circle Distance (0/3)			
535 - Globetrotter	★	Lev 4	--- ? ---
10075 - Airlines	★	Lev 4	--- ? ---
11817 - Tunnelling the Earth	★	Lev 5	--- ? ---
Polygons (0/3)			
634 - Polygon	★	Lev 3	--- ? ---
10078 - The Art Gallery	★	Lev 3	--- ? ---
11447 - Reservoir logs	★	Lev 5	--- ? ---
Other Basic Geometry (0/3)			
10088 - Trees on My Island	★	Lev 4	--- ? ---
10297 - Beavergnaw	★	Lev 3	--- ? ---
11507 - Bender B. Rodríguez Problem	★	Lev 4	--- ? ---
Graham's Scan for Convex Hull (0/3 = 0%)			
Graham's Scan (0/3)			
109 - SCUD Busters	★	Lev 3	--- ? ---
811 - The Fortified Forest	★	Lev 5	--- ? ---
10065 - Useless Tile Packers	★	Lev 3	--- ? ---
Intersection Problems (0/6 = 0%)			
Line Segment Intersection (0/3)			
191 - Intersection	★	Lev 3	--- ? ---
378 - Intersecting Lines	★	Lev 3	--- ? ---
10902 - Pick-up Sticks	★	Lev 4	--- ? ---
Other Objects (0/3)			
460 - Overlapping Rectangles	★	Lev 3	--- ? ---
737 - Gleaming the Cubes	★	Lev 4	--- ? ---
10301 - Rings and Glue	★	Lev 3	--- ? ---
Divide and Conquer Revisited (0/3 = 0%)			
Divide and Conquer Revisited (0/3)			
10245 - The Closest Pair Problem	★	Lev 3	--- ? ---
10566 - Crossed Ladders	★	Lev 3	--- ? ---
11646 - Athletics Track	★	Lev 4	--- ? ---

270 - Lining Up

Time limit: 3.000 seconds



``How am I ever going to solve this problem?" said the pilot.

Indeed, the pilot was not facing an easy task. She had to drop packages at specific points scattered in a dangerous area. Furthermore, the pilot could only fly over the area once in a straight line, and she had to fly over as many points as possible. All points were given by means of integer coordinates in a two-dimensional space. The pilot wanted to know the largest number of points from the given set that all lie on one line. Can you write a program that calculates this number?

Your program has to be efficient!

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.
The input consists of N pairs of integers, where $1 < N < 700$. Each pair of integers is separated by one blank and ended by a new-line character. The list of pairs is ended with an end-of-file character. No pair will occur twice.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.
The output consists of one integer representing the largest number of points that all lie on one line.

Sample Input

```
1

1 1
2 2
3 3
9 10
10 11
```

Sample Output

```
3
```

10263 - Railway

Time limit: 3.000 seconds

Problem A: Railway

Problem

Railway is a broken line of *N* segments. The problem is to find such a position for the railway station that the distance from it to the given point *M* is the minimal.

Input

The input will consist of several input blocks. Each input block begins with two lines with coordinates *Xm* and *Ym* of the point *M*. In the third line there is *N* - the number of broken line segments. The next *2N+2* lines contain the *X* and the *Y*coordinates of consecutive broken line corners.

The input is terminated by <EOF>.

Output

For each input block there should be two output lines. The first one contains the first coordinate of the station position, the second one contains the second coordinate. Coordinates are the floating-point values with four digits after decimal point.

Sample Input

```
6
-3
3
0
1
5
5
9
-5
15
3
0
0
1
1
0
2
0
```

Sample Output

```
7.8966
-2.2414
1.0000
0.0000
```

11227 - The silver bullet.

Time limit: 3.000 seconds

Time limit: 1s

Your blood-thirsty ``friend" Gnuffalo Bill wants to set the new world record to be published in the A.H.A. (American Hunters Association) Records Book. His idea is to use the brand new Barata gun to kill as many gnus as possible with a single shot. What a nice fellow Bill is! Well, he has not so kindly asked you to help him figure out, given a map of positions of gnus, how many he can kill at once. To avoid his wrath, you rush to your work table and start writing a program to fulfill Bill's request.

Input

The first line of input gives the number of cases, T ($1 \leq T \leq 10$). T test cases follow. Each one contains a number N, such that $100 \geq N \geq 1$, followed by Nlines describing the positions of the gnus. Each such line contains two numbers, with two decimal values, describing the cartesian coordinates X and Y of a gnu, such that $100.00 \geq X, Y \geq -100.00$. Positions may be repeated, but only count once.

Output

The output is comprised of one line for each input data set. The line identifies the data set with a number (starting from one and incrementing at each new data set), the number of gnus and the maximum number of aligned gnus. The exact format shall follow the sample output.

Sample input

```
3
5
0.00 0.00
0.00 0.00
1.00 1.00
1.00 0.00
0.00 1.00
2
0.00 0.00
0.00 0.00
6
0.00 2.00
0.00 0.00
1.00 1.00
1.00 0.00
0.00 1.00
0.00 -2.00
```

Sample output

```
Data set #1 contains 4 gnus, out of which a maximum of 2 are aligned.
Data set #2 contains a single gnu.
Data set #3 contains 6 gnus, out of which a maximum of 4 are aligned.
```

Problem setter: David Deharbe.
Universidade Federal do Rio Grande do Norte Qualifying Contest IV, June 9th, 2007.

10005 - Packing polygons

Time limit: 3.000 seconds

Packing polygons

Given a polygon of *n* points (not necessarily convex), your goal is to say whether there is a circle of a given a radius *R* that contains the polygon or not.

Input

The input consists of several input cases. The first line of each input case is the number *n* (with $n < 100$) of vertices in the polygon. Then you are given *n* lines each containing a couple of integers that define the coordinates of the vertices. The last line of the input case will be a real number indicating the radius *R* of the circle.

The end of the input will be signaled by an input case with $n = 0$ vertices, that must not be processed.

You may assume that no vertex will appear twice in any given input case.

Output

If the polygon can be packed in a circle of the given radius you have to print:

```
The polygon can be packed in the circle.
```

If the polygon cannot be packed you have to print:

```
There is no way of packing that polygon.
```

Sample Input

```
3
0 0
1 0
0 1
1.0
3
0 0
1 0
0 1
0.1
0
```

Sample Output

```
The polygon can be packed in the circle.
There is no way of packing that polygon.
```

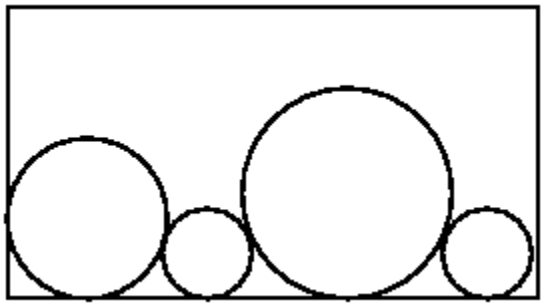
Miguel Revilla
2000-08-21

10012 - How Big Is It?

Time limit: 3.000 seconds

How Big Is It?

Ian's going to California, and he has to pack his things, including his collection of circles. Given a set of circles, your program must find the smallest rectangular box in which they fit. All circles must touch the bottom of the box. The figure below shows an acceptable packing for a set of circles (although this may not be the optimal packing for these particular circles). Note that in an ideal packing, each circle should touch at least one other circle (but you probably figured that out).



Input

The first line of input contains a single positive decimal integer *n*, *n*<=50. This indicates the number of lines which follow. The subsequent *n* lines each contain a series of numbers separated by spaces. The first number on each of these lines is a positive integer *m*, *m*<=8, which indicates how many other numbers appear on that line. The next *m* numbers on the line are the radii of the circles which must be packed in a single box. These numbers need not be integers.

Output

For each data line of input, excluding the first line of input containing *n*, your program must output the size of the smallest rectangle which can pack the circles. Each case should be output on a separate line by itself, with three places after the decimal point. Do not output leading zeroes unless the number is less than 1, e.g. `0.543`.

Sample Input

```
3
3 2.0 1.0 2.0
4 2.0 2.0 2.0 2.0
3 2.0 1.0 4.0
```

Sample Output

```
9.657
16.000
12.657
```

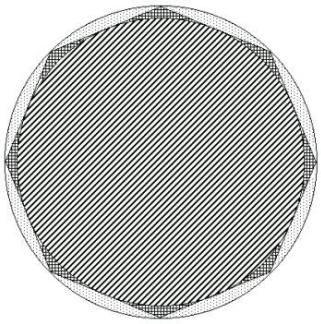
Miguel Revilla
2000-08-22

10451 - Ancient Village Sports

Time limit: 3.000 seconds

Time Limit: 30 seconds

In an ancient village there lived a number of people who liked different sorts of sports very much. But there was no built-in sports-ground. So they started looking for places. Ultimately what they found were the surfaces of the rocks with a shape of a regular polygon. Since the spectators are the part and parcel of the game, they must have some place over the ground. Again, officials of the teams must also have some space. They chose the in circle or striped portion of the regular polygon as the playing ground. They decided to leave the dotted portion as indicated in the figure for spectators and the criss-crossed portion for the officials. The diagonally striped portion is used for playing the game. You are going to help them to find the area of these portions.



A polygon is**regular** if all its sides are equal and all its angles are equal. Either of the conditions implies the other in the case of a triangle, but not in general. A rhombus has equal sides but not necessarily equal angles, and a rectangle has equal angles but not necessarily equal sides. So rhombus and rectangle are not regular polygons.

You are given the area (A) of an n-sided regular polygon. You are to determine the total area for the spectators and the total area for the officials. Assume that $\pi = 2*\cos^{-1}(0)$

Input

In each line of the input file there is an integer *n* (0<n<=50) and a floating-point number A (0<=A<=30000). A line with the value of n is less than three, terminates the input.

Output

For each line of input (except the last one) you should produce one line of output. This line contains the serial no of output as shown in the sample output followed by two floating-point numbers separated by a single space. The first one gives the area for the spectators and the second one gives that of the officials. There is also a single space between the colon and first floating point number. The floating-point numbers has five digits after the decimal point.

Sample Input

```
3 0.43301

6 2.59808

9 6.18182

0 2.33333
```

Sample Output

```
Case 1: 0.61418 0.17121

Case 2: 0.54352 0.24188

Case 3: 0.53226 0.25314
```

Problem-setter: Tanbir Ahmed, CSE Department, Southeast University

190 - Circle Through Three Points

Time limit: 3.000 seconds

Your team is to write a program that, given the Cartesian coordinates of three points on a plane, will find the equation of the circle through them all. The three points will not be on a straight line.

The solution is to be printed as an equation of the form

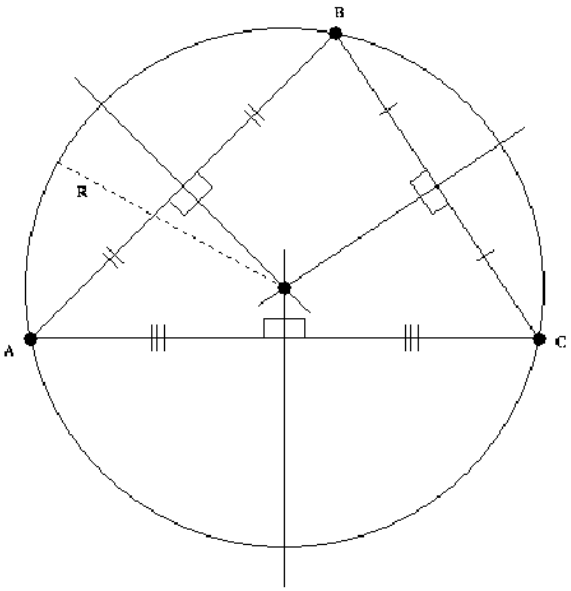
$$(x-h)^2+(y-k)^2=r^2 \qquad (1)$$

and an equation of the form

$$x^2+y^2+cx+dy+e=0 \qquad (2)$$

Each line of input to your program will contain the *x* and *y* coordinates of three points, in the order A_x, A_y, B_x, B_y, C_x, C_y. These coordinates will be real numbers separated from each other by one or more spaces.

Your program must print the required equations on two lines using the format given in the sample below. Your computed values for *h*, *k*, *r*, *c*, *d*, and *e* in Equations 1 and 2 above are to be printed with three digits after the decimal point. Plus and minus signs in the equations should be changed as needed to avoid multiple signs before a number. Plus, minus, and equal signs must be separated from the adjacent characters by a single space on each side. No other spaces are to appear in the equations. Print a single blank line after each equation pair.



Sample input

```
7.0 -5.0 -1.0 1.0 0.0 -6.0
1.0 7.0 8.0 6.0 7.0 -2.0
```

Sample output

```
(x - 3.000)^2 + (y + 2.000)^2 = 5.000^2
x^2 + y^2 - 6.000x + 4.000y - 12.000 = 0

(x - 3.921)^2 + (y - 2.447)^2 = 5.409^2
x^2 + y^2 - 7.842x - 4.895y - 7.895 = 0
```

10286 - Trouble with a Pentagon

Time limit: 3.000 seconds

Problem D
Trouble with a Pentagon
Input: standard input
Output: standard output
Time Limit: 2 seconds
Memory Limit: 32 MB

You are asked to place the largest possible square inside a regular pentagon (whose internal angles are same and all the sides are same in length). You are given the information that one vertex of the square will be coincident with a vertex of the square as shown in the figure below. You will have to find the length of a side of the square when a side of the regular pentagon is given.

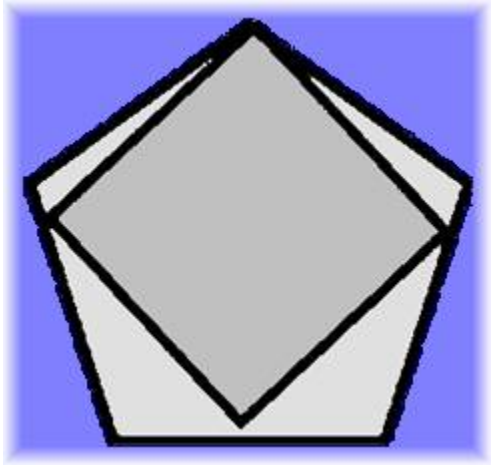


Fig: Square in a pentagon.

Input
The input file contains several lines of input. Each line contains a floating point number **F (0<=F<=100000)** which indicates the length of a side of the pentagon. Input is terminated by end of file.

Output
For each line of input produce one line of output containing a floating point number with ten digits after the decimal point. This number indicates the largest possible side of a square that fits in the pentagon. This output will be judged with a special correction program, so don't worry about small precision errors.

Sample Input
```
0.0000001
0.0000002
0.0000003
```

Sample Output
```
0.0000001067
0.0000002135
0.0000003202
```

(World Finals Warm-up Contest, Problem Setter: Shahriar Manzoor)

11152 - Colourful Flowers

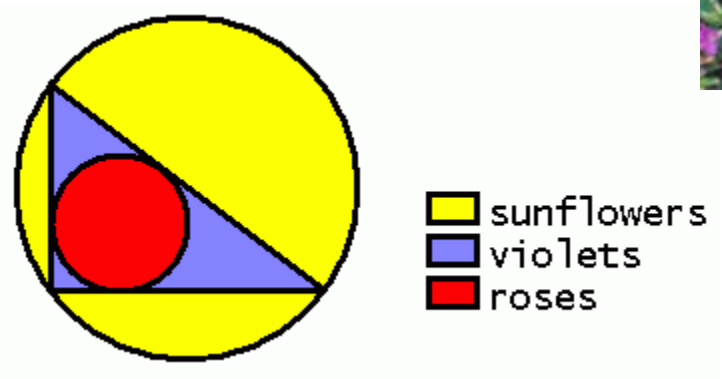
Time limit: 3.000 seconds

Problem E: Colourful Flowers

Time limit: 10 seconds

"Roses are red, violets are blue..."

Millionaire Mr Smith is well-known -- not for his wealth, but for his odd sense of "art"... Mr Smith has got a circular garden. On the boundary he picks three points and gets a triangle. He then finds the largest circle in that triangular region. So he gets something like this (Please click here for a black-and-white version of the figure):



Mr Smith then plants yellow sunflowers, blue violets and red roses in the way shown in the figure. (Nice combination, eh? :-) Given the lengths of the three sides of the triangle, you are to find the areas of the regions with each kind of flowers respectively.

Input and Output

Each line of input contains three integers a, b, c, the lengths of the three sides of the triangular region, with $0 < a \leq b \leq c \leq 1000$.

For each case, your program should output the areas of the regions with sunflowers, with violets and with roses respectively. Print your answers correct to 4 decimal places.

Sample Input

```
3 4 5
```

Sample Output

```
13.6350 2.8584 3.1416
```

Problemsetter: Mak Yan Kei

476 - Points in Figures: Rectangles

Time limit: 3.000 seconds

Points in Figures: Rectangles

Given a list of rectangles and a list of points in the *x*-*y* plane, determine for each point which figures (if any) contain the point.

Input

There will be $n(\leq 10)$ rectangles descriptions, one per line. The first character will designate the type of figure (``r" for rectangle). This character will be followed by four real values designating the *x*-*y* coordinates of the upper left and lower right corners.

The end of the list will be signalled by a line containing an asterisk in column one.

The remaining lines will contain the *x*-*y* coordinates, one per line, of the points to be tested. The end of this list will be indicated by a point with coordinates 9999.9 9999.9; these values should not be included in the output.

Points coinciding with a figure border are not considered inside.

Output

For each point to be tested, write a message of the form:

```
Point i is contained in figure j
```

for each figure that contains that point. If the point is not contained in any figure, write a message of the form:

```
Point i is not contained in any figure
```

Points and figures should be numbered in the order in which they appear in the input.

Sample Input

```
r 8.5 17.0 25.5 -8.5
r 0.0 10.3 5.5 0.0
r 2.5 12.5 12.5 2.5
*
2.0 2.0
4.7 5.3
6.9 11.2
20.0 20.0
17.6 3.2
-5.2 -7.8
9999.9 9999.9
```

Sample Output

```
Point 1 is contained in figure 2
Point 2 is contained in figure 2
Point 2 is contained in figure 3
Point 3 is contained in figure 3
Point 4 is not contained in any figure
```

```
Point 5 is contained in figure 1
Point 6 is not contained in any figure
```

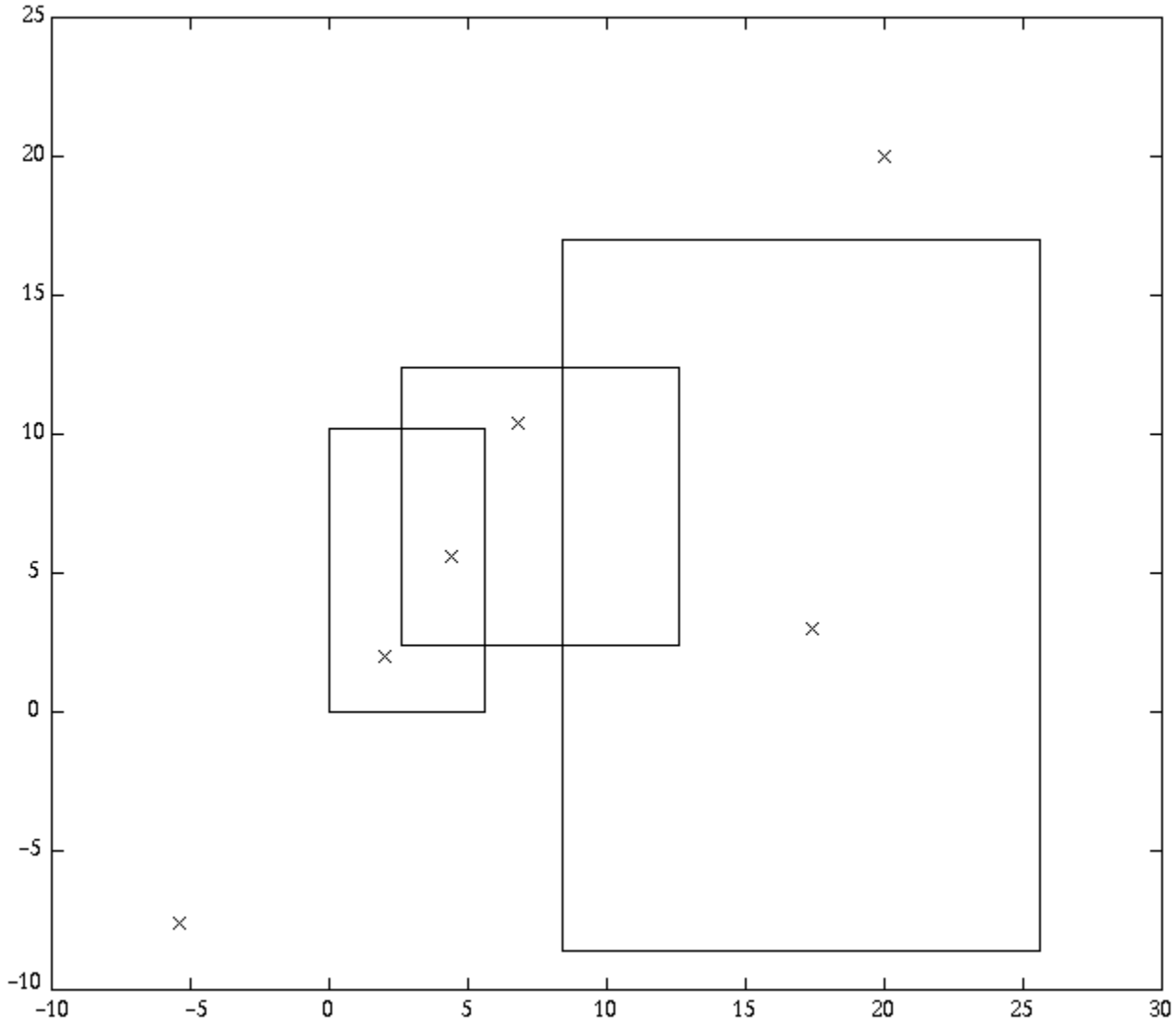
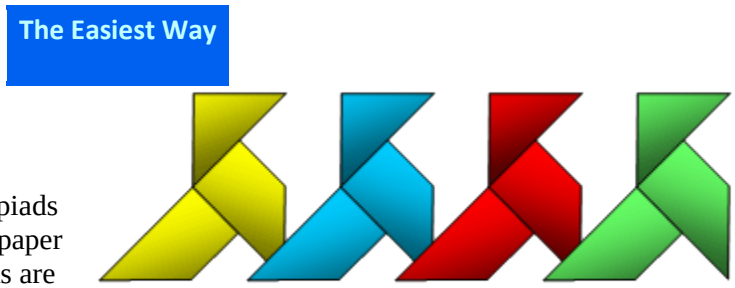


Diagrama of sample input figures and data points

11207 - The easiest way

Time limit: 3.000 seconds



Background

As a participant in the Programming Olympiads in Murcia, your purpose is to win as many paper birds as possible. But this year the problems are so difficult... So you decide to take the easiest way: to make the paper birds yourself.

The Problem

You want to make 4 paper birds of the same size, pretending that you have solved 4 problems --thus classifying for SWERC'2007--. N rectangular pieces of paper of different sizes are available. Each piece of paper, i, has width w_i and height h_i. Your task is to select one piece in order to maximize the size of the 4 birds. You have to take into account that, to make a bird, a square piece of paper is needed. The paper can be cut off, but not glued.

If more than one optimum piece of paper is possible, you have to indicate the first one.

The Input

The input may contain several test cases. For each test case, the first line indicates the number of pieces of paper, N. The following N lines contain the sizes of the pieces of paper; each line contains two integers: w_i and h_i. The input ends with a case where N=0.

The Output

For each test case (except for the case with N=0), the output should consist of an integer indicating the number of the piece to make the biggest paper birds. The first piece is 1, the second is 2, and so on. If many solutions are possible, output the first one.

Sample Input

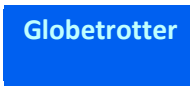
```
3
10 20
40 8
12 12
3
140 122
122 140
100 170
2
120 170
71 500
0
```

Sample Output

```
2
1
2
```

535 - Globetrotter

Time limit: 3.000 seconds



As a member of an ACM programming team you'll soon find yourself always traveling around the world: Zürich, Philadelphia, San José, Atlanta,... from 1999 on the Contest Finals even will be on a different continent each year, so one day you might get to Japan or Australia.

At the contest site it would be interesting to know how many miles you are away from home. For this sake, your job is to write a program to compute the geographical distance between two given locations on the Earth's surface.

We assume that the Earth is a perfect sphere with a radius of exactly 6378 km. The geographical distance between A and B is the length of the geodetic line segment connecting A and B.

The geodetic line segment between two points on a sphere is the shortest connecting curve lying entirely in the surface of the sphere.

The value of pi (π) is approximately 3.141592653589793.

Input Specification

The input file will consist of two parts: a list of cities and a list of queries.

City List

The city list consists of up to 100 lines, one line per city. Each line will contain a string c_i and two real numbers lat_i and $long_i$, representing the city name, its latitude and its longitude, respectively.

The city name will be shorter than 30 characters and will not contain white-space characters. The latitude will be between -90 (South Pole) and +90 (North Pole). The longitude will be between -180 and +180 where negative numbers denote locations west of the meridian and positive numbers denote locations east of the meridian. (The meridian passes through Greenwich, London.)

The city list will be terminated by a line consisting of a single ``#".

Query List

Each line will contain two city names *A* and *B*.

The query list will be terminated by the line ``# #".

Output Specification

For each query, print a line saying ``*A* - *B*" where *A* and *B* are replaced by the city names. Then print a line saying ``*x* km" where *x* is replaced by the geographical distance (in km) between the two cities, rounded to the nearest integer.

If one of the cities in the query didn't occur in the city list, print a line saying ``Unknown" instead.

Sample Input

```
Ulm 48.700 10.500
```

```
Freiburg 47.700 9.500
Philadelphia 39.883 -75.250
SanJose 37.366 -121.933
NorthPole 90 0
SouthPole -90 0
#
Ulm Philadelphia
Ulm SanJose
Freiburg Philadelphia
Freiburg SanJose
Ulm Freiburg
SanJose Philadelphia
Ulm LasVegas
Ulm Ulm
Ulm NorthPole
Ulm SouthPole
NorthPole SouthPole
# #
```

Sample Output

```
Ulm - Philadelphia
6536 km
Ulm - SanJose
9367 km
Freiburg - Philadelphia
6519 km
Freiburg - SanJose
9412 km
Ulm - Freiburg
134 km
SanJose - Philadelphia
4023 km
Ulm - LasVegas
Unknown
Ulm - Ulm
0 km
Ulm - NorthPole
4597 km
Ulm - SouthPole
15440 km
NorthPole - SouthPole
20037 km
```

10075 - Airlines

Time limit: 3.000 seconds

Problem A
Airlines
Input: standard input
Output: standard output

A leading airlines company has hired you to write a program that answers the following query: given a list of city locations (latitudes and longitudes) and a list of direct flights what is the minimum distance a passenger needs to fly to get from a given city to another?

To get from a city to another a passenger may either take a direct flight (if exists) or take a sequence of connecting flights (if there exists such a route).

Assume that if a passenger takes a direct flight from ***X*** to ***Y*** he never flies more than the geographical distance between ***X*** and ***Y***. The geographical distance between two locations ***X*** and ***Y*** is the length of the geodetic line segment connecting ***X*** and ***Y***. The geodetic line segment between two points on a sphere is the shortest connecting curve lying entirely in the surface of the sphere. Assume that the Earth is a perfect sphere with a radius of exactly 6378-km and the value of π is approximately **3.141592653589793**. Round the geographical distance between every pair of cities to the nearest integer.

Input
The input may contain multiple test cases.

The first line of each test case contains three integers **N** (N <= 100), **M** (M <= 300) and **Q** (Q <= 10000) where **N** indicates the number of cities, **M** represents the number of direct flights and **Q** is the number of queries.

The next **N** lines contain the city list. The **i**-th of these **N** lines will contain a string $\mathbf{c_i}$ followed by two real numbers $\mathbf{lt_i}$ and $\mathbf{ln_i}$, representing the city name, its latitude and longitude respectively. The city name will be no longer than 20 characters and will not contain white-space characters. The latitude will be between -90 (South Pole) and +90 (North Pole). The longitude will be between -180 and +180 where negative numbers denote locations west of the meridian and positive numbers denote locations east of the meridian. (The meridian passes through Greenwich, London.)

The next **M** lines contain the direct flight list. The **i**-th of these **M** lines will contain two city names $\mathbf{a_i}$ and $\mathbf{b_i}$ indicating that there exists a direct flight from city $\mathbf{a_i}$ to city $\mathbf{b_i}$. Be assured that both city names will occur in the city list.

The next **Q** lines contain the query list. The **i**-th of these **Q** lines will contain two city names $\mathbf{a_i}$ and $\mathbf{b_i}$ asking for the minimum distance a passenger needs to fly in order to get from city $\mathbf{a_i}$ to city $\mathbf{b_i}$. Be assured that $\mathbf{a_i}$ $\mathbf{b_i}$ are not equal and both city names will occur in the city list.

The input will terminate with three zeros form **N**, **M** and **Q**.

Output
For each test case in the input first output the test case number (starting from 1) as shown in the sample output. Then for each query in the input print a line giving the shortest distance (in km) a passenger needs to fly to get from the first city ($\mathbf{a_i}$) in the query to the second one ($\mathbf{b_i}$). If there exists no route form $\mathbf{a_i}$ to $\mathbf{b_i}$, just print the line "no route exists".

Print a blank line between two consecutive test cases.

Sample Input

```
3 4 2
Dhaka 23.8500 90.4000
Chittagong 22.2500 91.8333
Calcutta 22.5333 88.3667
```

```
Dhaka Calcutta
Calcutta Dhaka
Dhaka Chittagong
Chittagong Dhaka
Chittagong Calcutta
Dhaka Chittagong
5 6 3
Baghdad 33.2333 44.3667
Dhaka 23.8500 90.4000
Frankfurt 50.0330 8.5670
Hong_Kong 21.7500 115.0000
Tokyo 35.6833 139.7333
Baghdad Dhaka
Dhaka Frankfurt
Tokyo Hong_Kong
Hong_Kong Dhaka
Baghdad Tokyo
Frankfurt Tokyo
Dhaka Hong_Kong
Frankfurt Baghdad
Baghdad Frankfurt
0 0 0
```

Sample Output

```
Case #1
485 km
231 km

Case #2
19654 km
no route exists
12023 km
```

Rezaul Alam Chowdhury

11817 - Tunnelling the Earth

Time limit: 1.000 seconds

Problem E: Tunnelling the Earth

There are different methods of transporting people from place to place: cars, bikes, boats, trains, planes, etc. For very long distances, people generally fly in a plane. But this has the disadvantage that the plane must fly around the curved surface of the earth. A distance travelled would be shorter if the traveller followed a straight line from one point to the other through a tunnel through the earth.

For example, travelling from Waterloo to Cairo requires a distance of 9293521 metres following the great circle route around the earth, but only 8491188 metres following the straight line through the earth.

For this problem, assume that the earth is a perfect sphere with radius of 6371009 metres.

Input Specification

The first line of input contains a single integer, the number of test cases to follow. Each test case is one line containing four floating point numbers: the latitude and longitude of the origin of the trip, followed by the latitude and longitude of the destination of the trip. All of these measurements are in degrees. Positive numbers indicate North latitude and East longitude, while negative numbers indicate South latitude and West longitude.

Sample Input

```
1
43.466667 -80.516667 30.058056 31.228889
```

Output Specification

For each test case, output a line containing a single integer, the difference in the distance between the two points following the great circle route around the surface of the earth and following the straight line through the earth, in metres. Round the difference of the distances to the nearest integer number of metres.

Output for Sample Input

```
802333
```

Ondřej Lhoták

634 - Polygon

Time limit: 3.000 seconds

Modern graphic computer programs can, among other, even more stunning capabilities, fill a closed region. Though not all of them can protect the user from accidentally choosing to fill the background rather than the inner part. Besides being a channel hopper at home your boss' favourite hobby is colouring the pictures, you cannot protest long about adding this magnificent protection feature to his graphic program.

This means that your job is to write a program, which determines whether a point belong to a polygon, given the array of its vertices.

To make life a bit simpler you may assume that:

- all edges of the polygon are vertical or horizontal segments
- lengths of all the edges of the polygon are even integer numbers
- co-ordinates of at least one vertex are odd integer numbers
- both co-ordinates of any vortex cannot be divisible by 7 at the same time
- the investigated point P has both co-ordinates being even integer numbers
- the polygon has at most 1000 vertices
- co-ordinates of the vertices lay in the range: -10000..10000.

Input

Input data may consist of several data sets, each beginning with a number of polygon's vertices (*n*). Consecutive *n* lines contain co-ordinates of the vertices (*x*followed by *y*). Then go the co-ordinates of investigated point P. Input data end when you find 0 the number of polygon's vertices.

Output

For each polygon and each point P you should print one character (in separate lines): T when P belongs to the polygon or F otherwise.

Sample Input

```
4
1 1
1 3
3 3
3 1
2 2
12
1 1
1 9
3 9
3 5
5 5
5 9
7 9
7 1
5 1
5 3
3 3
3 1
4 2
0
```

Sample Output

```
T
F
```

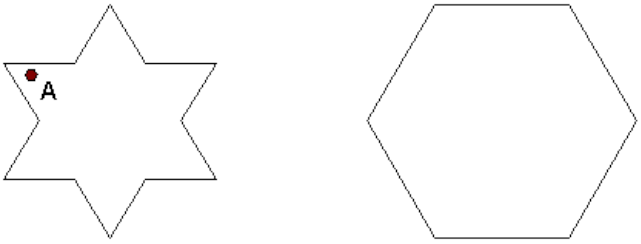
10078 - The Art Gallery

Time limit: 3.000 seconds

Problem D
The Art Gallery
Input: standard input
Output: standard output

Century Arts has hundreds of art galleries scattered all around the country and you are hired to write a program that determines whether any of the galleries has a *critical point*. The galleries are polygonal in shape and a *critical point* is a point inside that polygon from where the entire gallery is not visible.

For example, in gallery 1 (drawn below) point *A* is a critical point, but gallery 2 has no critical point at all.



Gallery 1 Gallery 2

The *Century Arts* authorities will provide you with the shape of a gallery as a sequence of (x, y) co-ordinates (determined using a suitable origin) of the consecutive corner points of that gallery.

Input
The input file consists of several data blocks. Each data block describes one gallery.

The first line of a data block contains an integer N ($3 \le N \le 50$) indicating the number of corner points of the gallery. Each of the next N lines contains two integers giving the (x, y) co-ordinates of a corner point where $0 \le x, y \le 1000$. Starting from the first point given in the input the corner points occur in the same order on the boundary of the gallery as they appear in the input. No three consecutive points are co-linear.

The input file terminates with a value of 0 for N.

Output
For each gallery in the input output the word "*Yes*" if the gallery contains a critical point, otherwise output the word "*No*". Each output must be on a separate line.

Sample Input
```
4
0 0
3 0
3 3
0 3
4
0 0
3 0
1 1
0 3
0
```

Sample Output
```
No
Yes
```

11447 - Reservoir logs

Time limit: 2.000 seconds

A. Reservoir Logs

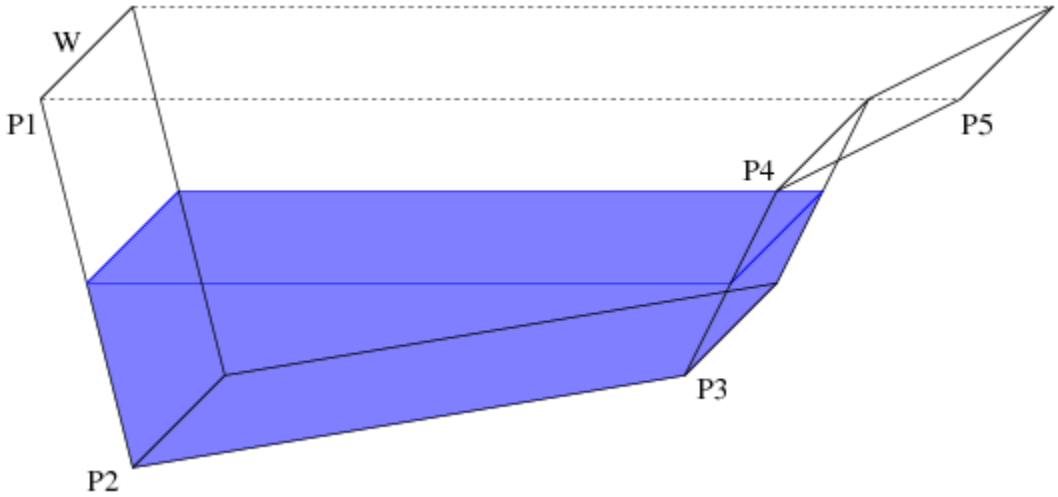
Background

Lack of rain in Spain is becoming a serious problem. A decrease in annual rainfall since the 1970s is said to have produced the present low reservoir levels. Although reservoirs across the country are running at an average capacity of less than 50 percent, in the worst affected areas such as Murcia, the capacity in many reservoirs is said to be less than 20 percent.

You are a scientist that must compute each month the capacity, in percentage, of water of a given reservoir.

The Problem

A reservoir is described by a list of coordinates ($P[1], P[2], ..., P[N]$) which define its cross section, and the width W of the reservoir, as shown in the figure below. The points that define the cross section always go from the left to the right, i.e., $P[i].x <= P[i+1].x$.



Cross section of the reservoir: Both dimensions and coordinates are given in metres.

We know the level of the reservoir at the beginning of the month (as a percentage). Also, we know the amount of water consumed due to evaporation and population use, and the amount of water falling on the reservoir because of rain (both measures are given in cubic metres). We suppose that water consumption only happens in the first half of the month, and the rain in the second half (so, in the same month, we can have a lack of water and then an excess of water).

The Input

The first line of the input contains an integer T ($0 < T < 200$), indicating the number of test cases. Test cases are separated by blank lines.

Each test case starts with a line containing the number N ($2 < N < 10000$) of coordinate pairs (x, y) of the reservoir, with $x >= 0$ and $y <= 0$. The following Nlines for each test case contain the coordinates defining the reservoir's cross section (two integer numbers per line). The first and last coordinates always have$y = 0$. Coordinates are always in order from left to right. The following line contains only an integer W indicating the width of the reservoir.

Finally, each test case ends with a line containing three real numbers representing: the initial percentage of water in the reservoir, the amount of spent water (in cubic metres), and the amount of water falling over the reservoir (also in cubic metres), respectively.

The Output

For each test case, you must write a line with the text "`Final percentage: X%`", where *X* is an integer number between 0 and 100 indicating the final capacity (as a percentage) of the reservoir; you have to take the integer part of the result (rounding down).

If the amount of water consumed is greater than the initial capacity of the reservoir, you must write the string "`Lack of water. `" before the previous text (observe that it ends with a blank space).

If the final amount of water exceeds the potential capacity of the reservoir, you must write "`Excess of water. `" (observe that it ends with a blank space) before the final percentage and, if it appears, after the "lack of water" text.

Note that in some cases the three texts may appear. Remember: first there is a water consumption and then water recovering by the rain.

Sample Input

```
3

4
0 0
0 -2
2 -2
2 0
1
50.0 2.0 1.0

9
0 0
9 -2
11 -1
16 -3
22 -3
23 0
28 -2
28 -4
36 0
2
84.2 178.24 991.00

3
0 0
1 -1
2 0
100
0.01 23.00 49.99
```

Sample Output

```
Final percentage: 25%
Lack of water. Excess of water. Final percentage: 100%
Lack of water. Final percentage: 49%
```

10088 - Trees on My Island

Time limit: 3.000 seconds

I have bought an island where I want to plant trees in rows and columns. So, the trees will form a rectangular grid and each of them can be thought of having integer coordinates by taking a suitable grid point as the origin.

But, the problem is that the island itself is not rectangular. So, I have identified a simple polygonal area inside the island with vertices on the grid points and have decided to plant trees on grid points lying strictly inside the polygon.

Now, I seek your help for calculating the number of trees that can be planted on my island.

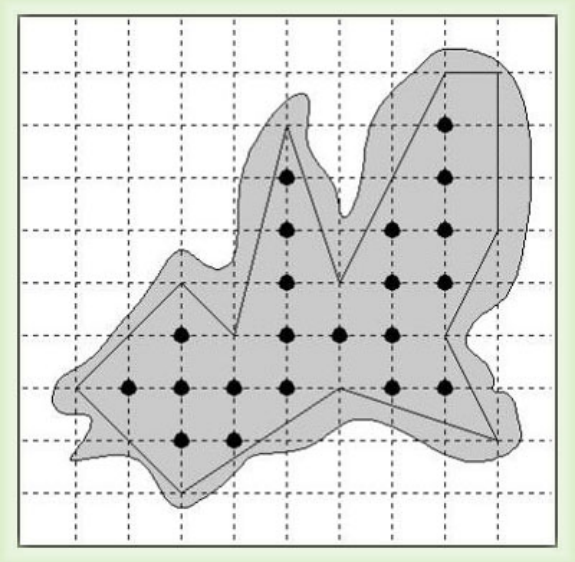


Figure: A sample of my island

Input
The input file may contain multiple test cases. Each test case begins with a line containing an integer N ($3 <= N <= 1{,}000$) identifying the number of vertices of the polygon. The next N lines contain the vertices of the polygon either in clockwise or in anti-clockwise direction. Each of these N lines contains two integers identifying the x and y-coordinates of a vertex. You may assume that none of the coordinates will be larger than 1,000,000 in absolute values.

A test case containing a zero for N in the first line terminates the input.

Output
For each test case in the input print a line containing the number of trees that can be planted inside the polygon.

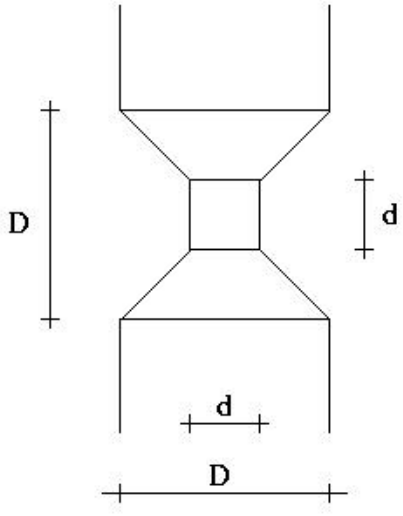
Sample Input
```
12
3 1
6 3
9 2
8 4
9 6
9 9
8 9
6 5
5 8
4 4
3 5
1 3
12
1000 1000
2000 1000
4000 2000
6000 1000
8000 3000
8000 8000
7000 8000
5000 4000
4000 5000
3000 4000
3000 5000
1000 3000
0
```

Sample Output
```
21
25990001
```

10297 - Beavergnaw
Time limit: 3.000 seconds

Problem C: Beavergnaw

When chomping a tree the beaver cuts a very specific shape out of the tree trunk. What is left in the tree trunk looks like two frustums of a cone joined by a cylinder with the diameter the same as its height. A very curious beaver tries not to demolish a tree but rather sort out what should be the diameter of the cylinder joining the frustums such that he chomped out certain amount of wood. You are to help him to do the calculations.



We will consider an idealized beaver chomping an idealized tree. Let us assume that the tree trunk is a cylinder of diameter **D** and that the beaver chomps on a segment of the trunk also of height **D**. What should be the diameter **d** of the inner cylinder such that the beaver chmped out **V** cubic units of wood?

Input contains multiple cases each presented on a separate line. Each line contains two integer numbers **D** and **V**separated by whitespace. **D** is the linear units and **V** is in cubic units. **V**will not exceed the maximum volume of wood that the beaver can chomp. A line with **D**=0 and **V**=0 follows the last case.

For each case, one line of output should be produced containing one number rounded to three fractional digits giving the value of **d**measured in linear units.

Sample input

```
10 250
20 2500
25 7000
50 50000
0 0
```

Output for sample input

```
8.054
14.775
13.115
30.901
```

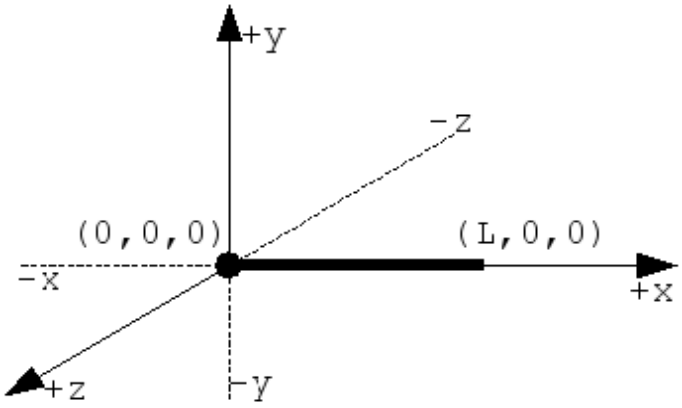
11507 - Bender B. Rodríguez Problem

Time limit: 4.000 seconds

Bender B. Rodríguez Problem

Bender is a robot built by *Mom's Friendly Robot Company* at its plant in Tijuana, Mexico in 2996 . He is a `Bending-Unit 22`, serial number 2716057 and chassis number 1729 . He was created for the task of bending metal wires.

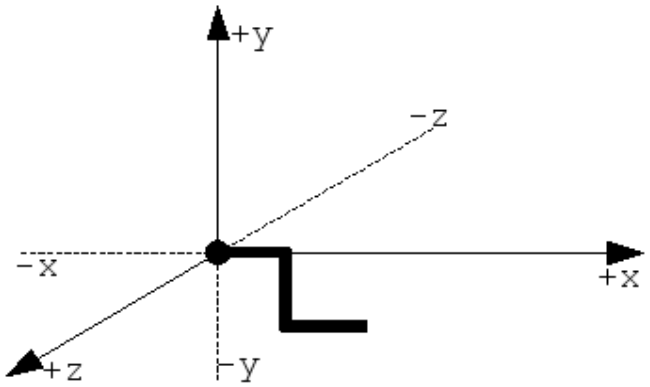
Bender needs to bend a wire of length `L` (`L` $\geq$ 2 an integer). The wire is represented in the Bender's brain (a `MOS Technology 6502` microprocessor) as a line stucked in the origin of a tridimensional cartesian coordinate system, and extended along the `X` positive axis (`+X`), so that the fixed extreme of the wire is in the coordinate `(0,0,0)` and the free extreme of the wire is in the coordinate `(L,0,0)`.



Bender bends the wire at specific points, starting at the point `(L-1,0,0)` and ending at the point `(1,0,0)`. For each `i` from `L-1` to `1`, Bender can take one of the following decisions:

- Not to bend the wire at point `(i,0,0)`.
- To bend the wire at point `(i,0,0)` an angle of $\frac{\pi}{2}$ to be parallel to the axis`+y`, `-y`, `+z` or `-z`.

For example, if `L=3` and Bender bends the wire at `(2,0,0)` on the `+y` axis direction, and at `(1,0,0)` on the `-y` axis direction, the result would be:



Given a sequence of bends, you must determine what direction is pointed by the last segment of the wire (`+X` in the example). You can suppose that the wire can intercept itself, after all it is the future!

Input

The first line of each test case gives an integer `L` ($2 \leq L \leq 100000$) indicating the length of the wire.

The second line of each test case contains the `L-1` decisions taken by Bender at each point, separated by spaces. The `j`-th decision in the list (for each $1 \leq j \leq L-1$) corresponds to the decision taken at the point `(L-j,0,0)`, and must be one of the following:

- `No` if the wire isn't bended at point `(L-j,0,0)`.
- `+y` if the wire is bended at point `(L-j,0,0)` on the `+y` axis.
- `-y` if the wire is bended at point `(L-j,0,0)` on the `-y` axis.
- `+z` if the wire is bended at point `(L-j,0,0)` on the `+z` axis.
- `-z` if the wire is bended at point `(L-j,0,0)` on the `-z` axis.

The end of the input is indicated when `L=0`.

Output

For each case in the input, print one line with the direction pointed by the last segment of the wire, `+x`, `-x`, `+y`, `-y`, `+z` or `-z` depending on the case.

Sample Input

```
3
+z -z
3
+z +y
2
+z
4
+z +y +z
5
No +z No No
0
```

Sample Output

```
+x
+z
+z
-x
+z
```

Colombia'2008

109 - SCUD Busters

Time limit: 3.000 seconds

SCUD Busters

Background

Some problems are difficult to solve but have a simplification that is easy to solve. Rather than deal with the difficulties of constructing a model of the Earth (a somewhat oblate spheroid), consider a pre-Columbian flat world that is a 500 kilometer $\times$ 500 kilometer square.

In the model used in this problem, the flat world consists of several warring kingdoms. Though warlike, the people of the world are strict isolationists; each kingdom is surrounded by a high (but thin) wall designed to both protect the kingdom and to isolate it. To avoid fights for power, each kingdom has its own electric power plant.

When the urge to fight becomes too great, the people of a kingdom often launch missiles at other kingdoms. Each SCUD missile (Sanitary Cleansing Universal Destroyer) that lands within the walls of a kingdom destroys that kingdom's power plant (without loss of life).

The Problem

Given coordinate locations of several kingdoms (by specifying the locations of houses and the location of the power plant in a kingdom) and missile landings you are to write a program that determines the total area of all kingdoms that are without power after an exchange of missile fire.

In the simple world of this problem kingdoms do not overlap. Furthermore, the walls surrounding each kingdom are considered to be of zero thickness. The wall surrounding a kingdom is the minimal-perimeter wall that completely surrounds all the houses and the power station that comprise a kingdom; the area of a kingdom is the area enclosed by the minimal-perimeter thin wall.

There is exactly one power station per kingdom.

There may be empty space between kingdoms.

The Input

The input is a sequence of kingdom specifications followed by a sequence of missile landing locations.

A kingdom is specified by a number N ($3 \leq N \leq 100$) on a single line which indicates the number of sites in this kingdom. The next line contains the x and $y$coordinates of the power station, followed by N-1 lines of x, y pairs indicating the locations of homes served by this power station. A value of -1 for N indicates that there are no more kingdoms. There will be at least one kingdom in the data set.

Following the last kingdom specification will be the coordinates of one or more missile attacks, indicating the location of a missile landing. Each missile location is on a line by itself. You are to process missile attacks until you reach the end of the file.

Locations are specified in kilometers using coordinates on a 500 km by 500 km grid. All coordinates will be integers between 0 and 500 inclusive. Coordinates are specified as a pair of integers separated by white-space on a single line. The input file will consist of up to 20 kingdoms, followed by any number of missile attacks.

The Output

The output consists of a single number representing the total area of all kingdoms without electricity after all missile attacks have been processed. The number should be printed with (and correct to) two decimal places.

Sample Input

```
12
3 3
4 6
4 11
4 8
10 6
5 7
6 6
6 3
7 9
10 4
10 9
1 7
5
20 20
20 40
40 20
40 40
30 30
3
10 10
21 10
21 13
-1
5 5
20 12
```

Sample Output

```
70.50
```

A Hint

You may or may not find the following formula useful.

Given a polygon described by the vertices $v_0, v_1, \ldots, v_n$ such that $v_0 = v_n$, the signed area of the polygon is given by

$$a = \frac{1}{2} \sum_{i=1}^{n} (x_{i-1} y_i) - (x_i y_{i-1})$$

where the x, y coordinates of $v_i = (x_i, y_i)$; the edges of the polygon are from v_i to v_{i+1} for $i = 0 \ldots n-1$.

If the points describing the polygon are given in a counterclockwise direction, the value of *a* will be positive, and if the points of the polygon are listed in a clockwise direction, the value of *a* will be negative.

811 - The Fortified Forest

Time limit: 3.000 seconds

Once upon a time, in a faraway land, there lived a king. This king owned a small collection of rare and valuable trees, which had been gathered by his ancestors on their travels. To protect his trees from thieves, the king ordered that a high fence be built around them. His wizard was put in charge of the operation.

Alas, the wizard quickly noticed that the only suitable material available to build the fence was the wood from the trees themselves. In other words, it was necessary to cut down some trees in order to build a fence around the remaining trees. Of course, to prevent his head from being chopped off, the wizard wanted to minimize the value of the trees that had to be cut. The wizard went to his tower and stayed there until he had found the best possible solution to the problem. The fence was then built and everyone lived happily ever after.

You are to write a program that solves the problem the wizard faced.

Input

The input contains several test cases, each of which describes a hypothetical forest. Each test case begins with a line containing a single integer n, $2 \leq n \leq 15$, the number of trees in the forest. The trees are identified by consecutive integers 1 to n. Each of the subsequent lines contains 4 integers x_i, y_i, v_i, l_i that describe a single tree. (x_i, y_i) is the position of the tree in the plane, v_i is its value, and l_i is the length of fence that can be built using the wood of the tree. $v_i$and l_i are between 0 and 10,000.

The input ends with an empty test case ($n = 0$).

Output

For each test case, compute a subset of the trees such that, using the wood from that subset, the remaining trees can be enclosed in a single fence. Find the subset with a minimum value. If more than one such minimum-value subset exists, choose one with the smallest number of trees. For simplicity, regard the trees as having zero diameter.

Display, as shown below, the test case numbers (1, 2, ...), the identity of each tree to be cut, and the length of the excess fencing (accurate to two fractional digits).

Display a blank line between test cases.

Sample Input

```
6
 0  0  8  3
 1  4  3  2
 2  1  7  1
 4  1  2  3
 3  5  4  6
 2  3  9  8
3
 3  0 10  2
 5  5 20 25
 7 -3 30 32
0
```

Sample Output

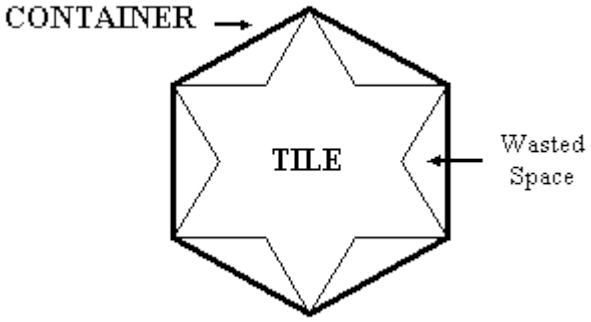
```
Forest 1
Cut these trees: 2 4 5
Extra wood: 3.16

Forest 2
Cut these trees: 2
Extra wood: 15.00
```

10065 - Useless Tile Packers

Time limit: 3.000 seconds

Yes, as you have apprehended the *Useless Tile Packers* (UTP) pack tiles. The tiles are of uniform thickness and have simple polygonal shape. For each tile a container is custom-built. The floor of the container is a convex polygon and under this constraint it has the minimum possible space inside to hold the tile it is built for. But this strategy leads to wasted space inside the container.



The UTP authorities are interested to know the percentage of wasted space for a given tile.

Input

The input file consists of several data blocks. Each data block describes one tile.

The first line of a data block contains an integer *N* (3 <= N <= 100) indicating the number of corner points of the tile. Each of the next *N* lines contains two integers giving the (*x*, *y*) co-ordinates of a corner point (determined using a suitable origin and orientation of the axes) where 0 <= x, , *y* <= *1000.*Starting from the first point given in the input the corner points occur in the same order on the boundary of the tile as they appear in the input. No three consecutive points are co-linear.

The input file terminates with a value of 0 for *N*.

Output

For each tile in the input output the percentage of wasted space rounded to two digits after the decimal point. Each output must be on a separate line. Print a blank line after each output block.

Sample Input

```
5
0 0
2 0
2 2
1 1
0 2
5
0 0
0 2
1 3
2 2
2 0
0
```

Sample Output

```
Tile #1
Wasted Space = 25.00 %

Tile #2
Wasted Space = 0.00 %
```

191 - Intersection

Time limit: 3.000 seconds

Intersection

You are to write a program that has to decide whether a given line segment intersects a given rectangle.

An example:

```
Line          : start point:(4,9)        end point:    (11,2)

Rectangle     : left-top:   (1,5)        right-bottom: (7,1)
```

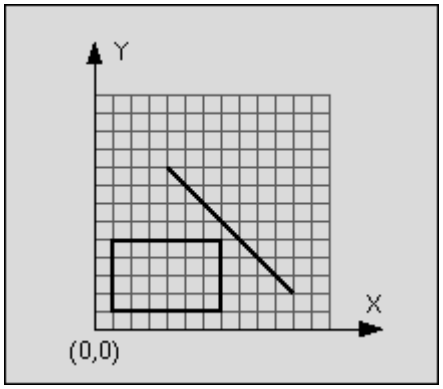


Figure: Line segment does not intersect rectangle

The line is said to intersect the rectangle if the line and the rectangle have at least one point in common. The rectangle consists of four straight lines and the area in between. Although all input values are integer numbers, valid intersection points do not have to lay on the integer grid.

Input

The input consists of *n* test cases. The first line of the input file contains the number *n*. Each following line contains one test case of the format:

xstart ystart xend yend xleft ytop xright ybottom

where (*xstart*, *ystart*) is the start and (*xend*, *yend*) the end point of the line and (*xleft*, *ytop*) the top left and (*xright*, *ybottom*) the bottom right corner of the rectangle. The eight numbers are separated by a blank. The terms *top left* and *bottom right* do not imply any ordering of coordinates.

Output

For each test case in the input file, the output file should contain a line consisting either of the letter "T" if the line segment intersects the rectangle or the letter "F" if the line segment does not intersect the rectangle.

Sample Input

```
1
4 9 11 2 1 5 7 1
```

Sample Output

```
F
```

378 - Intersecting Lines

Time limit: 3.000 seconds

Intersecting Lines

We all know that a pair of distinct points on a plane defines a line and that a pair of lines on a plane will intersect in one of three ways: 1) no intersection because they are parallel, 2) intersect in a line because they are on top of one another (i.e. they are the same line), 3) intersect in a point. In this problem you will use your algebraic knowledge to create a program that determines how and where two lines intersect.

Your program will repeatedly read in four points that define two lines in the *x*-*y*plane and determine how and where the lines intersect. All numbers required by this problem will be reasonable, say between -1000 and 1000.

Input

The first line contains an integer *N* between 1 and 10 describing how many pairs of lines are represented. The next *N* lines will each contain eight integers. These integers represent the coordinates of four points on the plane in the order $x_1 y_1 x_2 y_2 x_3 y_3 x_4 y_4$. Thus each of these input lines represents two lines on the plane: the line through (x_1, y_1) and (x_2, y_2) and the line through (x_3, y_3) and (x_4, y_4). The point (x_1, y_1) is always distinct from (x_2, y_2). Likewise with (x_3, y_3) and (x_4, y_4).

Output

There should be *N*+2 lines of output. The first line of output should read`INTERSECTING LINES OUTPUT`. There will then be one line of output for each pair of planar lines represented by a line of input, describing how the lines intersect: none, line, or point. If the intersection is a point then your program should output the *x* and *y* coordinates of the point, correct to two decimal places. The final line of output should read ``END OF OUTPUT''.

Sample Input

```
5
0 0 4 4 0 4 4 0
5 0 7 6 1 0 2 3
5 0 7 6 3 -6 4 -3
2 0 2 27 1 5 18 5
0 3 4 0 1 2 2 5
```

Sample Output

```
INTERSECTING LINES OUTPUT
POINT 2.00 2.00
NONE
LINE
POINT 2.00 5.00
POINT 1.07 2.20
END OF OUTPUT
```

10902 - Pick-up Sticks

Time limit: 3.000 seconds

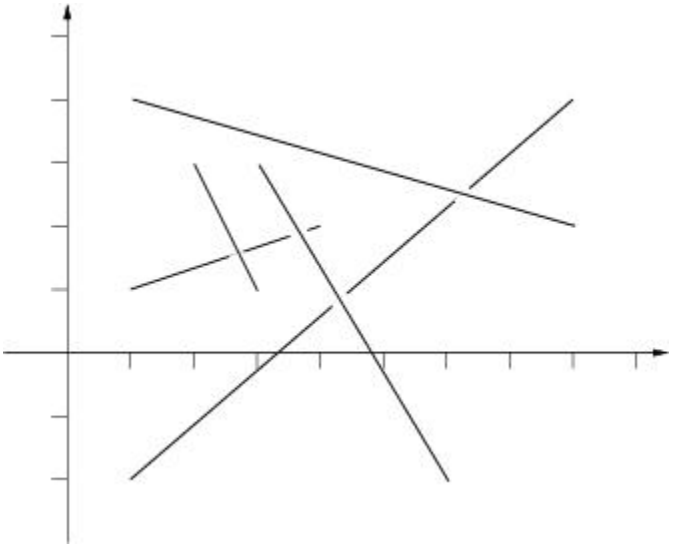
Problem C: Pick-up sticks

Stan has *n* sticks of various length. He throws them one at a time on the floor in a random way. After finishing throwing, Stan tries to find the top sticks, that is these sticks such that there is no stick on top of them. Stan has noticed that the last thrown stick is always on top but he wants to know all the sticks that are on top. Stan sticks are very, very thin such that their thickness can be neglected.

Input consists of a number of cases. The data for each case start with $1 \le n \le 100000$, the number of sticks for this case. The following *n* lines contain four numbers each, these numbers are the planar coordinates of the endpoints of one stick. The sticks are listed in the order in which Stan has thrown them. You may assume that there are no more than 1000 top sticks. The input is ended by the case with *n=0*. This case should not be processed.

For each input case, print one line of output listing the top sticks in the format given in the sample. The top sticks should be listed in order in which they were thrown.

The picture to the right below illustrates the first case from input.



Sample input

```
5
1 1 4 2
2 3 3 1
1 -2.0 8 4
1 4 8 2
3 3 6 -2.0
3
0 0 1 1
1 0 2 1
2 0 3 1
0
```

Output for sample input

```
Top sticks: 2, 4, 5.
Top sticks: 1, 2, 3.
```

Piotr Rudnicki

460 - Overlapping Rectangles

Time limit: 3.000 seconds

Overlapping Rectangles

When displaying a collection of rectangular windows on a SUN screen, a critical step is determining whether two windows overlap, and, if so, where on the screen the overlapping region lies.

Write a program to perform this function. Your program will accept as input the coordinates of two rectangular windows. If the windows do not overlap, your program should produce a message to that effect. If they do overlap, you should compute the coordinates of the overlapping region (which must itself be a rectangle).

All coordinates are expressed in "pixel numbers", integer values ranging from 0 to 9999. A rectangle will be described by two pairs of (*X*,*Y*) coordinates. The first pair gives the coordinates of the lower left-hand corner (*XLL*,*YLL*). The second pair gives the coordinates of the upper right-hand coordinates (*XUR*, *YUR*). You are guaranteed that *XLL* < *XUR* and *YLL* < *YUR*.

Input

Input will contain several test case. It begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

Each test case consists of two lines. The first contains the integer numbers XLL, YLL, XUR and YUR for the first window. The second contains the same numbers for the second window.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

For each set of input if the two windows do not overlap, print the message ``No Overlap". If the two windows do overlap, print 4 integer numbers giving the *XLL*,*YLL*, *XUR* and *YUR* for the region of overlap.

Note that two windows that share a common edge but have no other points in common are considered to have ``No Overlap".

Sample Input

```
1

0 20 100 120
80 0 500 60
```

Sample Output

```
80 20 100 60
```

737 - Gleaming the Cubes

Time limit: 3.000 seconds

Gleaming the Cubes

As chief engineer of the Starship Interprize, the task of repairing the hyperstellar, cubic, transwarped-out software has fallen on your shoulders. Simply put, you must compute the volume of the intersection of anywhere from 2 to 1000 cubes.

Input and Output

The input data file consists of several sets of cubes for which the volume of their intersections must be computed. The first line of the data file contains a number (from 2 to 1000) which indicates the number of cubes which follow, one cube per line. Each line which describes a cube contains four integers. The first three integers are the *x*, *y* and *z* coordinates of the corner of a cube, and the fourth integer is the positive distance which the cube extends in each of the three directions (parallel to the x, y, and z axes) from that corner.

Following the data for the first set of cubes will be a number which indicates how many cubes are in a second set, followed by the cube descriptions for the second set, again one per line. Following this will be a third set, and so on. Your program should continue to process sets of cubes, outputting the volume of their intersections to the output file, one set per line, until a zero is read for the number of cubes.

Note that the data file will always contain at least one set of cubes, and every set will contain at least 2 and at most 1000 cubes. For any given set of cubes, the volume of their intersections will not exceed 1,000,000 units.

Sample Input

```
2
0 0 0 10
9 1 1 5
3
0 0 0 10
9 1 1 5
8 2 2 3
0
```

Sample Output

```
25
9
```

Miguel Revilla
2000-08-31

10301 - Rings and Glue

Time limit: 3.000 seconds

Little John is in big trouble. Playing with his different-sized (and colored!) rings and glue seemed such a good idea. However, the rings now lay on the floor, glued together with some-thing that will definitely not come off with water. Surprisingly enough, it seems like no rings are actually glued to the floor, only to other rings. How about that!

You must help Little John to pick the rings off the floor before his mom comes home from work. Since the glue is dry by now, it seems like an easy enough task. This is not the case. Little John is an irrational kid of numbers, and so he has decided to pick up the largest component (most rings) of glued-together rings first. It is the number of rings in this largest component you are asked to find. Two rings are glued together if and only if they overlap at some point but no rings will ever overlap in only a single point. All rings are of the doughnut kind (with a hole in them). They can however, according to Little John, be considered "infinitely thin".

Input
Input consists of a number **(>0)** of problems. Each problem starts with the number of rings, **n**, where **0 ≤ n < 100**. After that, **n** rows follow, each containing a ring's physical attributes. That is, **3** floating point numbers, with an arbitrary number of spaces between them, describing the **x** coordinate and **y** coordinate for its center and its radius. All floating point numbers fit into the standard floating point type of your favorite programming language **(e.g., `float` for C/C++)**. Input ends with a single row with the integer **-1**.

Output
Output consists of as many grammatically correct answers as there were problems, each answer, on a separate line, being **'The largest component contains X ring(s).'** where **X** is the number of rings in the largest component.

Sample Input
```
4
0.0 0.0 1.0
-1.5 -1.5 0.5
1.5 1.5 0.5
-2.0 2.0 3.5
3
3.0 2.0 2.0
0.0 -0.5 1.0
0.0 0.0 2.0
5
-2.0 0.0 1.0
1.0 -1.0 1.0
0.0 1.0 0.5
2.0 0.0 1.0
-1.0 1.0 1.0
-1
```

Sample Output
```
The largest component contains 4 rings.
The largest component contains 2 rings.
The largest component contains 3 rings.
```

(Joint Effort Contest, Problem Source: Swedish National Programming Contest, arranged by department of Computer Science at Lund Institute of Technology.)

10245 - The Closest Pair Problem

Time limit: 3.000 seconds

Given a set of points in a two dimensional space, you will have to find the distance between the closest two points.

Input

The input file contains several sets of input. Each set of input starts with an integer **N (0<=N<=10000)**, which denotes the number of points in this set. The next **N** line contains the coordinates of **N** two-dimensional points. The first of the two numbers denotes the **X-coordinate** and the latter denotes the **Y-coordinate**. The input is terminated by a set whose **N=0**. This set should not be processed. The value of the coordinates will be less than **40000** and non-negative.

Output

For each set of input produce a single line of output containing a floating point number (with four digits after the decimal point) which denotes the distance between the closest two points. If there is no such two points in the input whose distance is less than **10000**, print the line **INFINITY**.

Sample Input

```
3
0 0
10000 10000
20000 20000
5
0 2
6 67
43 71
39 107
189 140
0
```

Sample Output

```
INFINITY
36.2215
```

(World Final Warm-up Contest, Problem setter: Shahriar Manzoor)

"Generally, a brute force method has only two kinds of reply, a) Accepted b) Time Limit Exceeded."

10566 - Crossed Ladders

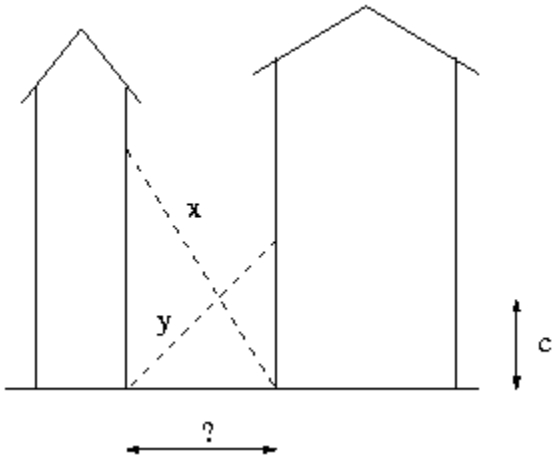
Time limit: 3.000 seconds

Problem H
Crossed Ladders
Input: Standard Input

Output: Standard Output

Time Limit: 1 Second

A narrow street is lined with tall buildings. An *x* foot long ladder is rested at the base of the building on the right side of the street and leans on the building on the left side. A *y* foot long ladder is rested at the base of the building on the left side of the street and leans on the building on the right side. The point where the two ladders cross is exactly *c* feet from the ground. How wide is the street?



Each line of input contains three positive floating point numbers giving the values of *x*, *y*, and *c*.

For each line of input, output one line with a floating point number giving the width of the street in feet, with three decimal digits in the fraction.

Sample Input	Output for Sample Input
30 40 10 12.619429 8.163332 3 10 10 3 10 10 1	26.033 7.000 8.000 9.798

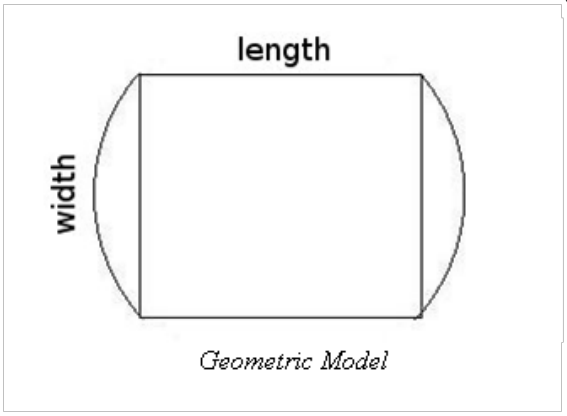
Problemsetter: Alberta Local Contest

11646 - Athletics Track

Time limit: 1.000 seconds

London Olympics is approaching very shortly – in just 3 years. Three years might not sound as that small a time to say 'just', but it is indeed for those who have to organize the competition. There are so many things to do – preparing the venues, building the Olympic village for accommodating athletes and

officials, improving the transportation of the entire city as the venues are located all over the city and also there will be great number of tourists / spectators during the Olympics.



Geometric Model


Actual View

One of the most important tasks is to build the stadium. You are appointed as a programmer to help things out in certain matters – more specifically in designing and building the athletics tracks. After some study, you find out that athletics tracks have a general shape of a rectangle with two sliced circles on two ends. Now the turf that is placed inside this rectangle is prepared elsewhere and comes in different shapes – different length to width ratios. You know one thing for certain – your track should have a perimeter of 400 meters. That's the standard length for athletics tracks. You are supplied with the design parameter – length to width ratio. You are also told that the sliced circles will be such that they are part of the same circle. You have to find the length and width of the rectangle.

Input

There will be at most 1000 test cases. Each test case will be given in one line. It will contain ratio of the length and width of the rectangle in the format – "a : b". Here, a and b will be integers and both will be between 1 and 1000 (inclusive).

Output

For each test case, output a line in the following format – "Case n: L W" where n is the case no (starting from 1) and L and W are length and width of the rectangle (in meters) respectively. You can output as many digits as you want after the decimal point. Output will be verified by a validator for 1E-5 precision.

Sample Input	Output for Sample Input
3 : 2 5 : 4	Case 1: 117.1858168913 78.1238779275 Case 2: 107.2909560477 85.8327648381