GEOMETRY PROBLEMS Fri Apr 1 05:00:14 EDT 2016

changeview
A computational view of the world.
Boston Preliminary 2003.

mirrors
Mirror, mirror, in the plane.
Boston Preliminary 2015

fractals
Fractionally increasing dimensions.
Boston Preliminary 2012

delaunay
Small circles are good circles.
Boston Preliminary 2011

xmole
Where is the beastie now?
Boston Preliminary 2015

reflect
Composition is such a wonder.
Boston Preliminary 2015

anglepuzzle
A protractor mess.
Boston Preliminary 2012

penrosetiling
Pretty but different.
Boston Preliminary 2010

```
Changing Point of View
-------- ----- -- ----

TeffalHead FatBody has stayed out too late on the planet
BadTrash and is in danger of being consumed by a Larger
BageGarLectorCol.  To get to safety TeffalHead must get
to base A or base B or the ZoomTube that connects them.
He knows his own position, C, and the ZoomTube is a
perfectly straight line between A and B (woe betide a
zoomer in a curved ZoomTube).  TeffalHead needs to know
immediately which he is closest to, A, B, or some point
on the ZoomTube between A and B.

TeffalHead knows the xy-coordinates of points A, B, and
C.  Like any good robotminded soul, he expects to trans-
late and rotate the xy-coordinate system to make a new
x'y'-coordinate system in which A has x'y'-coordinates
(0,0) and B has x'y'-coordinates (L,0), where L is the
distance from A to B.  Then the answer can be easily
read from the x' coordinate of C.

Unfortunately, living up to his first name, which means
'forgetful in emergencies', TeffalHead has forgotten the
program that finds the x'y'-coordinate system.  He as
put out a call for help, and as the only emergency prog-
grammer within range, you must send him a program tout
de suite.

Note you are permitted to translate and rotate the
xy-coordinates, but NOT to reflect across a coordinate
axis.  Unnecessary reflections are a terrible breech
of robot etiquette.  Thus the y' coordinate of C is
unambiguous.


Input
-----

For each of several cases, one line, containing

        Ax Ay Bx By Cx Cy

where the xy-coordinates of points A, B, and C are re-
spectively (Ax,Ay), (Bx,By), and (Cx,Cy).  Input ends
with an end of file.

Output
------

For each case one line containing:

        (Cx',Cy') L ANS

where (Cx',Cy') are the x'y'-coordinates of C, L is
the length of AB, and ANS is one of the following:

        A               If TeffalHead is closest to A.

        B               If TeffalHead is closest to B.

        ZoomTube        If TeffalHead is closest to a
                        point on the ZoomTube between
                        A and B.

The x'y'-coordinates and L must be accurate to plus or
minus 0.001.

Sample Input
------ -----

0 0 1 0 0.5 -6
5.0 3.0 5.5 2.5 5.0 4.0
5.0 3.0 5.5 2.5 5.0 1.0
```

```
Sample Output
------ ------

(0.500,-6.000) 1.000 ZoomTube
(-0.707,0.707) 0.707 A
(1.414,-1.414) 0.707 B



File:      changeview.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sun Oct 26 06:52:33 EST 2003

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.
```

Mirrors

Rosie the Robot has heard that every isometry of a plane can be constructed by at most 3 reflections. She’s not sure what this means, but she’s determined to find out. Her first thought is to find out what a reflection is.

She finds that a reflection about a straight line L in the plane is a map taking a point (x,y) in the plane to a point (x’,y’) such that

    (1) (x’,y’) == (x,y) if and only if (x,y) is on L

    (2) if (x,y) is NOT on L, the straight line between
        (x,y) and (x’,y’) is perpendicular to L and
        L bisects this line

Rosie decides to see how this works by computing (x’,y’) given (x,y) and L for some examples. The next question is how to specify L. She decides to do this by giving two points, (x1,y1) and (x2,y2) on L.

Your task is to write the program Rosie will use to compute (x’,y’) given (x,y), (x1,y1), (x2,y2).

The following picture visualizes the situation:

```
              (x,y)
                |
                |      (x1,y1)             (x2,y2)
    L ----------+---------*-------------------*------
                |
                |
             (x’,y’)
```

If L is a mirror and you are on the same side as (x,y) looking at L and seeing (x,y) reflected in the mirror, then this mirror reflection of (x,y) will appear to be at (x’,y’).

Input

For each of several test cases, a line containing just the test case name, followed by a line containing

    x1 y1 x2 y2

giving the points (x1,y1), (x2,y2) that specify the line L, followed by one or more lines each containing

    x y

giving a point (x,y) that is to be reflected about L, followed by a line containing just ‘*’.

All numbers are decimals with no more than 3 decimal places and an absolute value not greater than 10. No line will be longer than 80 characters. Input ends with an end of file.

```
Output
------

For each test case, first an exact copy of the test case
name line, then a line of the form

    x1  y1  x2  y2

that is the same as the test case second input line
except generally with a different number of decimal
places and different spacing, and then for each x y
test case input line one line of the form:

    x  y  x'  y'

repeating x and y and giving the point (x',y') that
is (x,y) reflected about L.  Output for the test case
ends with a line containing just '*'.

Each number output should take exactly 8 columns and
have exactly 3 decimal places.
```

```
Sample Input
------ -----

-- X-AXIS --
0 0 1 0
0 0
0 1
1 0
-1 0
0 -1
*
-- HORIZONTAL LINE --
0 1 1 1
0 0
0 1
1 0
-1 0
0 -1
-4 -10
*
-- 45 DEGREE DIAGONAL --
0 0 1 1
0 0
0 1
1 0
-1 0
0 -1
*
-- ARCTAN 3/4 = 36.86989765 DEGREE DIAGONAL --
0 0 4 3
0 0
0.5 0
-0.333 0.667
0.25 9.125
-7.359 8.004
*
```

```
Sample Output
------ ------

-- X-AXIS --
   0.000   0.000   1.000   0.000
   0.000   0.000   0.000   0.000
   0.000   1.000   0.000  -1.000
   1.000   0.000   1.000   0.000
  -1.000   0.000  -1.000   0.000
   0.000  -1.000   0.000   1.000
*
-- HORIZONTAL LINE --
   0.000   1.000   1.000   1.000
   0.000   0.000   0.000   2.000
   0.000   1.000   0.000   1.000
   1.000   0.000   1.000   2.000
  -1.000   0.000  -1.000   2.000
   0.000  -1.000   0.000   3.000
  -4.000 -10.000  -4.000  12.000
*
-- 45 DEGREE DIAGONAL --
   0.000   0.000   1.000   1.000
   0.000   0.000   0.000   0.000
   0.000   1.000   1.000   0.000
   1.000   0.000   0.000   1.000
  -1.000   0.000  -0.000  -1.000
   0.000  -1.000  -1.000  -0.000
*
-- ARCTAN 3/4 = 36.86989765 DEGREE DIAGONAL --
   0.000   0.000   4.000   3.000
   0.000   0.000   0.000   0.000
   0.500   0.000   0.140   0.480
  -0.333   0.667   0.547  -0.506
   0.250   9.125   8.830  -2.315
  -7.359   8.004   5.623  -9.306
*
```

```
File:      mirrors.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sat Oct  3 03:02:44 EDT 2015
```

Fractals

One way of making fractals is to take a line drawing and repeatedly perform a self-similar replacement operation on the line segments. A self-similar operation is one that is invariant under scaling, rotation, and translation (and sometimes reflection).

To be specific, the line segment replacement operation is defined by giving the line segments that will replace a unit line segment. The sequence of line segments that replace the unit line segment is called a ‘generator’. Any line segment can be made by scaling, rotating, and translating the unit line segment, so the replacement of any line segment L can be calculated by scaling, rotating, and translating the generator in the same way that the unit line segment was scaled, rotated, and translated to make L. When scaling, all dimensions must be scaled equally, and reflections are not allowed.

The line segments are directed; each has a beginning and an end, and these CANNOT be switched.

To be even more specific, suppose the unit line segment

```
                (0,0) - (1,0)
```

is replaced by the generator

```
    (0,0) - (1/3,0) - (1/2,sqrt(3)/6) - (2/3,0) - (1,0)
```

This is the ‘Koch generator’, and consists of dividing the unit line segment into thirds, constructing an equilateral triangle with the middle third as a side, and erasing the middle third of the original line.

To apply the Koch generator to the line segment

```
        L = (5,-2) - (7,0)
```

we first make L from the unit segment by scaling the unit line segment by sqrt(8), then rotating counter-clockwise by 45 degrees, and lastly translating by (5,-2). Then we do the same to the Koch generator, and get 4 replacement line segments for L. See the first sample below.

Fractals are made by applying generator defined replacements to all the line segments in a line drawing, and then repeating this entire operation an infinite number of times. You have been asked to do this, but just a finite number of times.

The Koch generator is oriented. This means that if a line drawing segment has its beginning and ending switched, its Koch generator generated replacement will be a reflection of the original replacement. Specifically, the triangle will move to the other side of the original line. An oriented generator cannot sensibly be applied to a line drawing whose lines are not orientable. Some generators a not oriented, and can be applied to any line drawing.

Note also that the Koch generator is a connected curve from (0,0) to (1,0), but this is not required; a generator can be any set of line segments, possibly disjoint and possibly intersecting.

Input

For each of several test cases, first a line containing the test case name. Then one or more lines of the format

```
        x1 y1 x2 y2
```

each describing one line segment (x1,y1) - (x2,y2) of the generator that replaces (0,0) - (1,0). Then a line containing just '*' to signal the end of the generator. Next, more lines of the above format describing the line segments of the line drawing, followed by another line containing just '*'. The test case ends with a line containing just a single integer N, specifying the number of iterations of the replacement operation.

The generator will contain between 1 and 100 line segments, the line drawing will contain between 1 and 100 line segments, N will be between 0 and 10, and no line will longer than 80 characters.

Input ends with an end of file.

Output

For each test case, first a copy of the test case name line, then lines describing the line segments resulting from N replacements, and then a line containing just '*'. Each line that describes a line segment has the same format as in the input, and the numbers output in the line must be accurate to least 3 decimal places.

Initially the line drawing segments input are the current line segments, and these are in an ordered sequence. One iteration replaces EACH current line segment in order by the generator defined replacement. ORDER MUST BE MAINTAINED. There are N iterations. Note that N == 0 is possible (usable to display the input: see next paragraph).

The output may be printed as a graph or displayed in an X-window by the commands:

```
        print_fractals
        display_fractals
```

provided the output of your program has been stored in the file 'fractals.out'. To see the sample output instead use the commands

```
        print_fractals sample.test
        display_fractals sample.test
```

(here sample.test is the output for sample.in).

```
Sample Input
------ -----

-- KOCH CURVE; scale sqrt(8); rotate 45 deg --
0 0 0.3333333333 0
0.3333333333 0 0.5 0.28867513
0.5 0.28867513 0.6666666667 0
0.6666666667 0 1 0
*
5.00000000 -2.00 7 0
*
1
-- KOCH CURVE; 4 iterations --
0 0 0.3333333333 0
0.3333333333 0 0.5 0.28867513
0.5 0.28867513 0.6666666667 0
0.6666666667 0 1 0
*
0 0 1 0
*
4
-- KOCH FLAKE; 0 iterations --
[See sample.in file for rest of input]
```

```
Sample Output
------ ------

-- KOCH CURVE; scale sqrt(8); rotate 45 deg --
5.000 -2.000 5.667 -1.333
5.667 -1.333 5.423 -0.423
5.423 -0.423 6.333 -0.667
6.333 -0.667 7.000 0.000
*
-- KOCH CURVE; 4 iterations --
0.000 0.000 0.012 0.000
0.012 0.000 0.019 0.011
0.019 0.011 0.025 0.000
0.025 0.000 0.037 0.000
0.037 0.000 0.043 0.011
0.043 0.011 0.037 0.021
0.037 0.021 0.049 0.021
[see sample.test file for rest of output]
```

Reference

See Chapter 1 of ‘Fractals, Chaos, and Power Laws’ by Manfred Schroeder.

The formal definition of a ‘fractal’ is ‘a set whose fractal dimension exceeds its topological dimension’. However, the term ‘fractal dimension’ refers to one of many not exactly equivalent ways of computing dimension. If we use generators that are connected curves and apply them an infinite number of times we can generate a fractal whose topological dimension is 1. If we use Hausdorff dimension and the Koch generator the fractal dimension is log(4)/log(3). There are many other ways of generating fractals of topological dimension 1.

File: fractals.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Wed Oct 10 04:20:40 EDT 2012

```
Delaunay Triangulation
-------- -------------

You have been asked to find the Delaunay Triangulation
of a set S of points in the plane.

The Delaunay Triangulation of a set S of points in the
plane is a triangulation of the convex hull of S such
that the circumcircle of each triangle has no points of
S in its interior.  As long as there is no circle with 4
or more points of S on its boundary and no points of S
in its interior, the Delaunay Triangulation of S is
unique, and the edges of the triangulation are just the
edges of triangles with vertices in S which have no
points of S in the interior of their circumcircle.

The Delaunay Triangulation of S is coveted because among
all the possible triangulations of S it is the one that
maximizes the minimum angle between edges of the triang-
ulation.


Input
-----

For each of several test cases, first a line containing
nothing but the name of the test case, and then lines
containing the numbers

    N x[1] y[1] x[2] y[2] ... x[N] y[N]

where (x[i],y[i]) is the i'th point of S for 1 <= i <=
N.  3 <= N <= 100.  The xy coordinates are floating
point.

To simplify things, the input will be such that the
Delaunay triangulation is unique; that is, no 4 points
of S will be on the same circle if that circle contains
no points of S in its interior.

Input ends with an end of file.


Output
------

For each test case, first a line that is an exact copy
of the test case name input line.  Then one line for
each edge of the Delaunay Triangulation of S, this line
having the format

        i j

to specify that there is an edge from (x[i],y[i]) to
(x[j],y[j]).  Here 1 <= i,j <= N.  Do NOT output any
edge more than once.

The output may be printed as a graph or displayed in an
X-window by the commands:

        print_graph
        display_graph

provided the input and output of your program has been
stored in the files

        delaunay.in
        delaunay.out

and the test case name lines in these files do not have
a digit as their first non-whitespace character.  To see
the sample output instead use the commands

        print_graph sample.in sample.test
        display_graph sample.in sample.test

(here sample.test is the output for sample.in).
```

Note: The relative neighbor graph computed in the Relative Neighbor Graphs problem is a subgraph of the Delaunay Triangulation.

```
Sample Input
------ -----

-- SAMPLE 1 --
3 1 4 3 2 5 8
-- SAMPLE 2 --
7 -1.01 0 -1.01 5 1.01 2.01 3.04 3.02
  5.05 2.003 8.21 0 8.22 5.03


Sample Output
------ ------

-- SAMPLE 1 --
1 2
2 3
1 3
-- SAMPLE 2 --
1 2
2 3
1 3
3 5
1 5
5 6
1 6
3 4
2 4
4 7
2 7
4 5
5 7
6 7
```

File: delaunay.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Mon Oct 3 05:59:33 EDT 2011

X-Mole

Es Ophagus lives in the X-plane, which is 2D. She is having a terrible problem with an X-mole in her X-yard and needs to find it so she can X it out.

She’s rented an X-mole finder and hooked it up to her computer. When she presses a button, the finder generates a random line intersecting her yard, and tells her which side of the line the X-mole is on. It presents this information as 3 numbers, a, b, and c, such that

```
        a*x + b*y <= c
```

where (x,y) are the coordinates of the X-mole.

The finder generates a, b, and c at random, tests that the line a*x + b*y = c intersects the yard and tries again if not, and then if the line intersects the yard, outputs a, b, and c if the above equation is true, and -a, -b, and -c otherwise.

However, if the X-mole is so near the line the finder is not sure which side it is on, the finder discards the line and tries another.

Es wants to have some idea after every button press where the X-mole might be, so after every press she wants her computer to give her a bounding rectangle in which the X-mole must be, using information from the the button press and all previous button presses to make this bounding rectangle as small as possible. She wants you to program the computer to output these bounding rectangles.

The yard is bounded by

```
        0 <= x <= 1,000
        0 <= y <= 1,000
```

And, oh yes, Es is X-tra fast and needs an X-tra precise location so there are X-tra many button presses.

Input

For each of several test cases, a line containing just the test case name, followed by lines containing

```
        a b c
```

for each button press, followed by a line containing just ‘*’ to indicate the test run is finished.

Input ends with an end of file.

No line will have more than 80 characters.

The a, b, and c numbers are such that

```
    (a,b) is a unit vector
    - 2,000 <= c <= 2,000
    a, b, c have 9 decimal places
```

The number of button presses in one test case will not exceed 1,000,000, and in one file will not exceed 10,000,000. But hey, its really random!

Output

For each test case, first an exact copy of the input test case name line, then for each button press that CHANGES the bounding rectangle, 5 numbers:

```
        n xmin xmax ymin ymax
```

where n is the number of the button press (1, 2, ...) that changed the rectangle, (xmin,ymin) is the lower left corner of the rectangle, and (xmax,ymax) is the upper right corner of the rectangle. Each number must take exactly 10 columns. n is an integer, but the other numbers must have exactly 3 decimal places.

After all these location lines output a line containing just ‘*’ to end the test case.

You may assume that the X-mole actually exists in the yard and that the X-mole finder works perfectly.

Note: Your output can be expanded to include lines for button presses that did not change the bounding rectangle. E.g., Sample 1 Output below can be expanded to

```
         1     0.000  1000.000     0.000  1000.000
         2     0.000  1000.000     0.000  1000.000
         3     0.000   141.421     0.000   141.421
         4     0.000   141.421     0.000   141.421
         5     7.071   141.421     0.000    70.711
         6     7.071    10.000     4.142    10.000
```

by adding a line for button press 2 that copies the rectangle limits from the previous line, and similarly for button press 4. The judge will expand your output before testing for correctness. However, if you produce already expanded output, it will generate an Output Size Limit Exceeded error.

Sample Input
------ -----

```
-- SAMPLE 1 --
 1.000000000  0.000000000 1000.000000000
 0.000000000  1.000000000 1000.000000000
 0.707106781  0.707106781  100.000000000
-0.707106781 -0.707106781  -10.000000000
-0.707106781  0.707106781    0.000000000
 1.000000000  0.000000000   10.000000000
*
-- SAMPLE 2 --
 0.600000000  0.800000000 1000.000000000
-0.800000000  0.600000000  100.000000000
-0.600000000 -0.800000000 -200.000000000
 0.800000000 -0.600000000  -50.000000000
*
```

Sample Output
------ ------

```
-- SAMPLE 1 --
         1     0.000  1000.000     0.000  1000.000
         3     0.000   141.421     0.000   141.421
         5     7.071   141.421     0.000    70.711
         6     7.071    10.000     4.142    10.000
*
-- SAMPLE 2 --
         1     0.000  1000.000     0.000  1000.000
         2     0.000  1000.000     0.000   860.000
         3    40.000  1000.000     0.000   860.000
         4    40.000   560.000   190.000   860.000
*
```

File: xmole.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Sun Oct 11 07:19:18 EDT 2015

Reflect

Every isometry of N-dimensional space can be represented as the composition of at most N+1 reflections. You are given isometries, and must find representations of each as a composition of at most N+1 reflections.

You are given isometries in the form

```
        v |---> T + Mv
```

where v and T are N-vectors and M is an NxN orthogonal matrix.

You must output equivalent representations of the form

```
        v |---> R[1]R[2]R[3]...R[K]v
```

where R[i] is a reflection across the hyperplane

```
        { v : U[i].v = C[i] },
```

U[i] is a unit vector, C[i] a real number, and U[i].v is the scalar product of U[i] and v, so that R[i] is characterized by U[i] and C[i]. It is required that K <= N+1 and that C[i] not be too large.

An isometry may have many different equivalent representations as such compositions of reflections, and you are being asked to output just one.

Input

For each of several test cases, a line containing just the test case name, followed by a line containing

```
    N T[1] T[2] ... T[N]
```

where T = (T[1], T[2], ..., T[N]) is the translation vector, followed by N lines containing M in the format

```
    M[1,1] M[1,2] ... M[1,N]
    M[2,1] M[2,2] ... M[2,N]
    ........................
    M[N,1] M[N,2] ... M[N,N]
```

The isometry is

```
    (T+Mv)[i] = T[i] + sum over j of M[i,j]*v[j]
```

or in other words, vectors are to be viewed as column vectors.

N is an integer, and the other numbers are floating point. 2 <= N <= 20. T[.] has an absolute value not greater than 10. Because M is an orthogonal matrix, M[.,.] cannot have an absolute value greater than 1. The test case name line is not longer than 80 characters, but other lines may be longer. Input ends with an end of file.

```
Output
------

For each test case, first an exact copy of the test case
name line, then a line containing just K, and then K
lines, the i'th representing R[i], of the form

        C[i] U[i][1] U[i][2] ... U[i][N]

In this line each number should take exactly 10 columns
and have exactly 6 decimal places.  U[i] must be a unit
vector and the absolute value of C[i] must not be
greater than 10*N.

The equivalence required is

        R[1]R[2]...R[K]v = T + Mv

The judge will check that this equation holds to 3 deci-
mal places for v equal to each of the N+1 points:

        (0,0,...,0)     [Origin]
        (1,0,...,0)     [N Unit Vectors]
        (0,1,...,0)
        ..........
        (0,0,...,1)

Here the judge is using the fact that if two isometries
agree on N+1 points, and the points span an N-dimension-
al affine subspace, the two isometries are identical on
that subspace.  This in turn follows from the fact that
any point P on the straight line through two distinct
points P1 and P2 is uniquely determined by its distances
from P1 and P2.

Solutions are not unique, you are to output any one.
You are NOT required to find a solution with a minimum
number of reflections.
```

```
Sample Input
------ -----

-- SAMPLE 1 --
2 0 0
0 -1
1 0
-- SAMPLE 2 --
2 1 0
0 -1
1 0
-- SAMPLE 3 --
2 1 0
0 1
1 0


Sample Output
------ ------

-- SAMPLE 1 --
2
  0.000000 -0.707107  0.707107
  0.000000  0.000000  1.000000
-- SAMPLE 2 --
2
  0.500000  1.000000  0.000000
  0.000000 -0.707107  0.707107
-- SAMPLE 3 --
3
  0.500000  1.000000  0.000000
  0.000000 -0.707107  0.707107
  0.000000  0.000000  1.000000
```

File: reflect.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Mon Aug 17 14:04:50 EDT 2015

```
Angle Puzzle
----- ------

An angle puzzle consists of a finite set of vertices in
the plane and a set of equations of the form

        xyz = some angle
    or
        xyz = ?

where x, y, and z are vertices and x != y != z != x.
xyz is interpreted as the angle at vertex y in the
triangle with vertices x, y, and z, if x, y, and z are
not all on the same infinite line, with this angle being
positive if traversing from x to y to z goes in the
counter-clockwise direction about the triangle, and
negative if clockwise.

But if x, y, and z are on the same infinite line, xyz =
0 if x and z are on the same side of y, and xyz = 180
if x and z are on opposite sides of y.  These are useful
ways of saying that x, y, and z are on the same line.

Note that xyz = - zyx always and adding multiples of
360 to an angle does not change the angle (so 180 and
-180 are equal as angles).  All input and output angles
are measured in degrees and are in the range (-180,180],
so +180 is allowed for input/output but -180 is NOT
allowed and must be replaced by +180.

The puzzle requires you to solve for the ?'s in the
the equations.

Sample 1 below is actually solvable using elementary
geometry without trigonometry, but in general you will
need to use trigonometry to solve these puzzles, as is
done in sample 2.


Input
-----

For each of several test cases, first a line with the
test case name, and then a sequence of lines with equa-
tions as above, and then a line containing just '.'.
The vertex names are all single capital letters.  The
angles are all in degrees.  The only space characters
in any input line other than the test case name line
are the two surrounding the '='.  No line is longer
than 80 characters.

No two vertices with different names are identical.

Input ends with an end of file.



Output
------

For each test case, a copy of the input but with ALL
'?'s replaced by numbers.  All output angles should have
exactly 3 decimal places and be in the range
(-180,+180].  The output should be an exact copy of the
input except for the replacement of '?'s by the numbers
and the rounding of input angles to 3 decimal places.

This problem is actually open ended in that we do not
expect you to find all the angles that might be
determined from the given input.  But you must find the
angles you are asked to find.  These can be found by
using only non-trigonometric constraints on angles plus
trigonometrically computed relative positions of the
vertices of any triangle two of whose angles are known.
```

```
Sample Input
------ -----

-- SAMPLE 1 --
ABC = 60.000000000
BCA = 60.000000000
DBC = 30.000000000
ADC = 180
DAB = ?
ADB = ?
.
-- SAMPLE 2 --
ABD = 60.000000000
DBC = 20.000000000
ADC = 180.000000000
EAB = 70.000000000
CAE = 10.000000000
CEB = 180.000000000
AED = ?
AEB = ?
EDB = ?
CBE = ?
.
```

```
Sample Output
------ ------

-- SAMPLE 1 --
ABC = 60.000
BCA = 60.000
DBC = 30.000
ADC = 180.000
DAB = 60.000
ADB = -90.000
.
-- SAMPLE 2 --
ABD = 60.000
DBC = 20.000
ADC = 180.000
EAB = 70.000
CAE = 10.000
CEB = 180.000
AED = 20.000
AEB = -30.000
EDB = 110.000
CBE = 0.000
.
```

File: anglepuzzle.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Wed Oct 10 02:54:56 EDT 2012

```
Penrose Tiling
------- ------

Sir Roger Penrose investigated aperiodic tilings of the
plane in the 1970's.  These tilings are generated
from a small number of finite shapes by following a set
of rules, but no translation of a tiling is identical to
the tiling, hence the designation 'aperiodic'.

Penrose rhombus tilings are generated from a pair of
rhombi called 't', the 'thin' rhombus, and 'T', the
'thick' rhombus.  All sides of these are unit length.
The angles of t are 36 and 144 degrees, and those of T
are 72 and 108.

The sides of the rhombi also need to be labeled, so
we give the following algorithms for drawing them using
a pen:

  for t, thin rhombus:
    draw a straight line of unit length labeled +1
    turn left 1*36 = 36 degrees
    draw a straight line of unit length labeled -1
    turn left 4*36 = 144 degrees
    draw a straight line of unit length labeled +2
    turn left 1*36 = 36 degrees
    draw a straight line of unit length labeled -2
    turn left 4*36 = 144 degrees
    you are now back in your starting position

  for T, thick rhombus:
    draw a straight line of unit length labeled +1
    turn left 2*36 = 72 degrees
    draw a straight line of unit length labeled +2
    turn left 3*36 = 108 degrees
    draw a straight line of unit length labeled -2
    turn left 2*36 = 72 degrees
    draw a straight line of unit length labeled -1
    turn left 3*36 = 108 degrees
    you are now back in your starting position

The rhombi must be fit together so:

    1. The rhombi are rotated and/or translated but NOT
       flipped over.

    2. Two rhombi may not intersect.  This means that
       their intersection as sets, including boundaries,
       must not contain any points EXCEPT for those in
       shared vertices and shared edges.

    3. When an edge is shared between two rhombi, the
       sum of the two labels of the edge must be 0.
       E.g., a +2 edge from one rhombus may be shared
       with a -2 edge from another rhombus, but NOT with
       a +2 or -1 or +1 edge.

    4. There are no holes in the tiling.

In this problem you are given a proposed finite Penrose
rhombic tiling and you are asked to determine whether
it follows all the above rules.

We need a way to describe a finite Penrose rhombic
tiling.  We do this by placing the tiles down on the
xy-plane so that each tile but the first shares an edge
with one of the tiles laid down so far.

The first tile is always a T-tile with its +1 edge dir-
ected from (0,0) to (1,0) and its +2 edge directed from
(1,0) to (x,y) with x>1, y>0.  This is referred to as
the 'standard position' for the first tile, which is
also tile 1 in a our tile labeling scheme that numbers
the n tiles laid down so far from 1 through n.
```

The position of the n+1'st tile is given by the line

```
        k j e
```

where

```
        k is the kind of tile, either 't' or 'T'
        j is the number of a previous tile that is to
          share an edge with the new tile; 1 <= j <= n
        e is the label (+1, -1, +2, or -2) of the edge
          of tile j that is to be shared with the new
          tile, respecting the rule about the sum of
          shared edge labels being zero
```

Thus the line 't 7 -2' says to lay a t-tile so that its +2 edge is shared with the -2 edge of the 7'th tile laid.

Input

The input consists of test cases. Each test case begins with a line containing the name of the test case. This is followed by any number of lines each containing a description 'k j e' of another tile to be laid to make a tiling pattern. The first tile of the pattern is in standard position, and the i'th line of the form 'k j e' describes how to lay the i+1'st tile. After these lines there is a line containing just '.', which is the last line of the test case.

```
        maximum number of tiles <= 10,000

        for each tile vertex (x,y):
            -100 <= x <= +100
            -100 <= y <= +100
```

Output

For each test case, first output an exact copy of the test case name line, and then output just one line in one of the following formats:

```
        tile # intersects tile #
        tile # edge # is shared with tile # edge #
        there are # holes
        tiling OK
```

Here the #'s are integers that are tile labels, edge labels, or counts. The first line is output if two tiles intersect; the second if two share edges have labels not summing to 0. If the tiling violates the rule against intersection AND the rule against edge labels not summing to 0, then either of the first two lines may be output -- only one violation is to be reported.

However, reporting holes must ONLY be done if there are NO intersection or edge label sum violations.

Printing Input
-------- -----

As a debugging aid, the command

```
        print_penrosetiling foo.in
```

will print a picture of the tiling described in foo.in. The file sample.ps contains the result for the sample input.

```
The labels in the picture are represented by single
arrows (+1, -1) or double arrows (+2, -2) going around
the rhombus boundary in the counter-clockwise (+1, +2)
or clockwise (-1, -2) directions.  They are offset so
that usually if a shared edge has labels not summing to
zero this will be visible in the picture.  But their
are perverse cases; consider:

        -- PERVERSE CASE --
        t 1 +1
        T 2 -1
        .


Sample Input
------ -----

This is available in the file sample.in.


Sample Output
------ ------

-- PENROSE TILING SAMPLE 1 --
tiling OK
-- PENROSE TILING SAMPLE 2 --
tile 1 and tile 7 intersect
```


```
File:       penrosetiling.txt
Author:     Bob Walton <walton@seas.harvard.edu>
Date:       Sun Nov 20 03:03:12 EST 2011
```