Problems Index Sun Oct 11 11:58:37 EDT 2015

BOSPRE 2015 PROBLEMS
------ ---- --------

The problems are in approximate order of difficulty, easiest first.

```
    problems/asciiart
        A picture is worth a thousand moves.
        Boston Preliminary 2015

    problems/modcalc
        Keeping arithmetic within bounds.
        Boston Preliminary 2015

    problems/vigenere
        Bicycle, Tricycle, wordy N-cycle.
        Boston Preliminary 2015

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

    problems/trends
        Luck or unluck tis common enough.
        Boston Preliminary 2015

    problems/xmole
        Where is the beastie now?
        Boston Preliminary 2015

    problems/latin
        Squares must be complete.
        Boston Preliminary 2015

    problems/reflect
        Composition is such a wonder.
        Boston Preliminary 2015

    problems/reversedraw
        Optimize the machine.
        Boston Preliminary 2015
```

ASCII Art
----- ---

ASCII art consists of a drawing made in an ASCII text file, such as

```
                     (
                  .-----.
                 /       \
                ,         ,
                |  O   O  |
                .    :    .
                 \       /
                  \  -  /
                   *---*
```

You are to write a program that creates ASCII art from an input command line.

Input

For each of several test cases, a line containing just the test case name, followed by a single command line from which the art is to be created.

Assume your printer has a head that begins at the bottom left of a blank page of ASCII characters and can print a single ASCII character or move one position. Then each command line character is interpreted as follows:

```
    n           move up (North) one square
    e           move right (East) one square
    s           move down (South) one square
    w           move left (West) one square
    any         print the character
    other
    character
```

For example, the command line ‘nn*es+’ is interpreted as move up, move up, print ‘*’, move right, move down, print ‘+’.

The page is 40 rows (lines) and 56 columns. There are two special cases. A request to move off the page is turned into a no-operation. A request to overwrite a character previously written in a page location is honored as if the previous character was never written.

Thus a command line that begins with ‘AwsB’ has the same effect as a command line that begins with just ‘B’, because the printer starts at the lower left so ‘w’ and ‘s’ become no-operations and ‘B’ overwrites ‘A’.

The command line may not be longer than 100,000 characters. Input ends with an end of file.

Output

For each test case, first an exact copy of the test case name line, and then the page with the drawing produced by the input command line.

However, blank lines at the TOP of the page MUST be omitted from the output.

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

-- SAMPLE 1 --
Aws*n|n|n*e-e-e-e-e-e*s|s|s*w-w-w-w-w-
-- SAMPLE 2 -
nnnn*ne/ne/ne*se\se\se*sw/sw/sw*nw\nw\nweeXeeXsw_
-- SAMPLE 3 -
[see sample.in]
-- SAMPLE 4 -
[see sample.in]
```

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

-- SAMPLE 1 --
*-----*
|     |
|     |
*-----*
-- SAMPLE 2 -
   *
  / \
 /   \
* X X *
 \ _ /
  \ /
   *

-- SAMPLE 3 -
     (
  .-----.
 /       \
'         '

|  O   O  |
.    :    .
 \       /
  \  -  /
   *---*
-- SAMPLE 4 -
[see sample.test: just '*' at 4 corners with many
 attempts to move off the page and with overwriting.]


File:      asciiart.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sun Oct 11 01:27:20 EDT 2015
```

```
Modulo Calculator
------ ----------

You have been asked to write a calculator for values
that are integers modulo m, for some integer m > 0.
As an initial experiment, you are to perform operations
on these integers and output the results, to see how
the operations behave.

Recall that if 0 <= x and 0 < m, then x modulo m is the
remainder of x divided by m.  For C, C++, and JAVA, it
is x % m.

The set { 0, 1, ..., m-1 } is sometimes denoted by [m].
You are to write a program that has input lines like

        [m] + n

which says add n to each i with 0 <= i < m (i.e., each
i in [m]) and output the result modulo m.  For example,
the line

        [5] + 1

outputs

        [5] + 1: 1 2 3 4 0

while the line

        [5] * 2

outputs

        [5] * 2: 0 2 4 1 3


Input
-----

Lines as follows:

    Line        Output for i, 0 <= i < m

    [m] + n     (i + n) modulo m [sum]
    [m] * n     (i * n) modulo m [product]
    [m] ** e    (i ** e) modulo m [exponential]
                Recall i ** 0 == 1 and i ** 1 == i.
    [m] -       The unique j such that 0 <= j < m and
                (i + j) == 0 modulo m.
    [m] /       The unique j such that 0 <= j < m and
                (i * j) == 1 modulo m, if such a j
                exists; otherwise output ‘*’.

In the above m, n, and e are integers with

    1 <= m <= 100    0 <= n < m    0 <= e <= 100

Input ends with an end of file.


Output
------

For each input line, a copy of the input line followed
by a colon and a space and then the outputs for each i,
i == 0 through i == m-1, in order.  Output lines for
larger m will be rather long.


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

[5] + 0
[5] + 1
[ see sample.in for rest of sample input ]
```

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

[5] + 0: 0 1 2 3 4
[5] + 1: 1 2 3 4 0
[5] + 2: 2 3 4 0 1
[5] * 0: 0 0 0 0 0
[5] * 1: 0 1 2 3 4
[5] * 2: 0 2 4 1 3
[5] ** 0: 1 1 1 1 1
[5] ** 1: 0 1 2 3 4
[5] ** 2: 0 1 4 4 1
[5] -: 0 4 3 2 1
[5] /: * 1 3 2 4
[10] ** 9: 0 1 2 3 4 5 6 7 8 9
[10] -: 0 9 8 7 6 5 4 3 2 1
[10] /: * 1 * 7 * * * 3 * 9
[11] + 6: 6 7 8 9 10 0 1 2 3 4 5
[11] + 7: 7 8 9 10 0 1 2 3 4 5 6
[11] * 2: 0 2 4 6 8 10 1 3 5 7 9
[11] * 3: 0 3 6 9 1 4 7 10 2 5 8
[11] * 9: 0 9 7 5 3 1 10 8 6 4 2
[11] * 10: 0 10 9 8 7 6 5 4 3 2 1
[11] ** 3: 0 1 8 5 9 4 7 2 6 3 10
[11] ** 8: 0 1 3 5 9 4 4 9 5 3 1
[11] ** 9: 0 1 6 4 3 9 2 8 7 5 10
[11] ** 99: 0 1 6 4 3 9 2 8 7 5 10
[11] ** 10: 0 1 1 1 1 1 1 1 1 1 1
[11] ** 100: 0 1 1 1 1 1 1 1 1 1 1
[11] -: 0 10 9 8 7 6 5 4 3 2 1
[11] /: * 1 6 4 3 9 2 8 7 5 10
[12] ** 10: 0 1 4 9 4 1 0 1 4 9 4 1
[12] ** 11: 0 1 8 3 4 5 0 7 8 9 4 11
[12] /: * 1 * * * 5 * 7 * * * 11
[20] /: * 1 * 7 * * * 3 * 9 * 11 * 17 * * * 13 * 19
[21] /: * 1 11 * 16 17 * * 8 * 19 2 * 13 * * 4 5 * 10 20
[22] /: * 1 * 15 * 9 * 19 * 5 * * * 17 * 3 * 13 * 7 * 21
```

```
NOTES:  For 0 <= i < m, it can be shown that there is
        AT MOST ONE j such that 0 <= j < m and i*j == 1
        modulo m.  If j exists, i is said to have a
        multiplicative inverse modulo m.

        For a given m, the set of i with a multiplica-
        tive inverse modulo m forms a group with multi-
        plication modulo m as its group operation.
        This group has applications in cryptography.
```


File: modcalc.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Fri Oct 2 14:54:11 EDT 2015

```
Vigenere Cipher
-------- ------

A Vigenere Cipher encrypts by shifting each letter of
a plain text line cyclically to the right within the
alphabet by an amount determined by a key.  For example,
'C' is 'A' shifted right 2 places and 'B' is 'Z' shifted
right 2 places cyclically (i.e., with wrap-around).

The key is made by repeating a keyword indefinitely.
The n'th letter of a plain text line is shifted
cyclically right by the same amount as the n'th letter
of the key is shifted right from the first letter A of
the alphabet.

So for example, if the keyword is CIPHER, the key is
CIPHERCIPHERCIPHER...., and the letters of a plain text
line are shifted cyclically right by the amounts:

    1st letter  shift by 2    (C is A shifted by 2)
    2nd letter  shift by 8    (I is A shifted by 8)
    3rd letter  shift by 15   (P is A shifted by 15)
    4th letter  shift by 7    (H is A shifted by 7)
    5th letter  shift by 4    (E is A shifted by 4)
    6th letter  shift by 17   (R is A shifted by 17)
    7th letter  shift by 2    (C is A shifted by 2)
    8th letter  shift by 8    (I is A shifted by 8)
    9th letter  shift by 15   (P is A shifted by 15)
   10th letter  shift by 7    (H is A shifted by 7)
   ...........  ..........    .....................

Thus for example, the line

        This is encrypted by a Vigenere Cipher.

is encrypted using the keyword 'CIPHER' to

        Vpxz zu tugiaxilh dg h Mkotuiig Rptygz.

because

        T --> V  (shift by 2)
        h --> p  (shift by 8)
        i --> x  (shift by 15)
        s --> z  (shift by 7)
          -->    (shift by 4 not used for space)
        i --> z  (shift by 17)
        s --> u  (shift by 2)
          -->    (shift by 8 not used for space)
        e --> t  (shift by 15)
        n --> u  (shift by 7)
        c --> g  (shift by 4)
        r --> i  (shift by 17 AND WRAP AROUND)
        etc.


Your spies have discovered an encrypted message, and
also the decrypted plain text of the first line of the
message.  They also have discovered that the message
lines were all encrypted using a Vigenere Cipher with
the same keyword, and that keyword has at most 10
letters.  You are to find the decrypted message.
```

```
Input
-----

For each of several test cases, a line containing just
the test case name, followed by the decrypted first
line of the message, followed by the encrypted message
which has one or more lines, followed by a line contain-
ing just ‘*’.

No line is longer than 80 characters and the message
has at most 40 lines.

Input ends with an end of file.


Output
------

For each test case, first an exact copy of the test case
name line, then the decrypted message, and lastly a line
containing just ‘*’.

The decrypted message is an exact copy of the encrypted
message but with letters changed.  Punctuation, spaces,
and line breaks are NOT changed.  The case of letters is
preserved (upper case to upper case, lower case to lower
case).

Test cases will be such that decryption is always
possible and is unique.
```

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

-- SAMPLE 1 --
This is encrypted by a Vigenere Cipher.
Vpxz zu tugiaxilh dg h Mkotuiig Rptygz.
Jq mscma,
    Lfrm fsl igl vprdfmei ioi ewcaijv.
Dgt jfnsh!
*
-- SAMPLE 2 --
AAAAAAAAAAAAAAAA
PASSWORDPASSWORD
PASSWORD
PASS
*
-- SAMPLE 3 --
Twas brillig, and the slithy toves
Kegz szosezo, tel aav ysbkpe mfdkz
Uqj zpzk tel nbdjrl zv aav chuv;
Rtr fzuyf nmxl kpk ufzunhmmy,
Rvj mym thdm ytkpy hlbmytsm.
*
```

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

-- SAMPLE 1 --
This is encrypted by a Vigenere Cipher.
Hi folks,
    Hope you are enjoying the contest.
Bye folks!
*
-- SAMPLE 2 --
AAAAAAAAAAAAAAAA
AAAAAAAA
AAAA
*
-- SAMPLE 3 --
Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.
*
```

File: vigenere.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Mon Oct 12 03:48:27 EDT 2015

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
```

```
Trends
------

Jack plays a solitary game in which he flips a coin and
gives himself a point on heads but removes a point on
tails.  He starts with a score of zero, and keeps score
as he flips coins.  You would think that his score would
not remain strictly positive for long, or strictly
negative for long, but you would be wrong.  Such long
periods of being strictly positive or strictly negative
are often mis-identified as trends that supposedly prove
the coin is biased.

You have been asked to find the probability of a trend
of length >= m in a sequence of n coin flips.  Here a
‘trend’ is defined as a sequence of consecutive flips
such that at the end of each flip Jack’s score is
strictly positive, or, at the end of each flip Jack’s
score is strictly negative.  For comparison purposes,
you are also asked to do this for both unbiased and
biased coins.


Input
-----

For each of several test cases, a line containing

        n m p

where n and m are as above and p is the probability of
heads.

 0 < m,n   m <= n   (m**2)*n <= 50,000,000   0 <= p <= 1

Input ends with an end of file.


Output
------

For each test case, a single line containing

        n m p ptrend

where n, m, p are copied from the input line and ptrend
is the probability of finding a trend of length >= m in
a sequence of n flips.  p and ptrend are to be printed
with exactly 3 decimal places (even if p is input with
fewer decimal places), whereas n and m are integers.


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

10 5 0.5
10 5 0.55
10 8 0.5
10 10 0.5
20 10 0.5
20 10 0.55
20 20 0.5

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

10 5 0.500 0.703
10 5 0.550 0.712
10 8 0.500 0.410
10 10 0.500 0.246
20 10 0.500 0.666
20 10 0.550 0.688
20 20 0.500 0.176
```

File: trends.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Thu Aug 13 06:26:42 EDT 2015

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

Latin Squares

A Latin square is an NxN matrix whose elements are the integers from 1 through N such that

* Each integer appears exactly once in each column.

* Each integer appears exactly once in each row.

A partial Latin square is an MxN matrix, with M rows and N columns, whose elements are integers from 1 through N, such that M < N and

* Each integer appears AT MOST once in each column.

* Each integer appears exactly once in each row.

Given any partial Latin square, it is possible to 'complete' it: that is, to add rows to its end to make a Latin square. You are being asked to write a program to do this.

Input

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

```
M N
```

followed by M lines containing an MxN partial Latin square in the form

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

Here L[r,c] is the element of the partial Latin square in row r and column c. All numbers are integers.

```
0 < M < N <= 1000
1 <= L[.,.] <= N
N*N*(N-M) <= 100,000,000
```

Output

For each test case, first an exact copy of the test case name line, and then N lines containing the completed Latin square:

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

Each element of the Latin square should be printed in 5 columns right adjusted. The first M lines should contain the partial Latin square that was the test case input.

There will be many possible correct answers: output only one.

Sample Input

```
-- SAMPLE 1 --
2 4
1 2 3 4
2 3 4 1
-- SAMPLE 2 --
1 10
1 2 3 4 5 6 7 8 9 10
```

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

-- SAMPLE 1 --
    1    2    3    4
    2    3    4    1
    3    4    1    2
    4    1    2    3
-- SAMPLE 2 --
    1    2    3    4    5    6    7    8    9   10
   10    9    8    6    7    5    4    3    2    1
    4   10    6    9    8    7    5    1    3    2
    6    5    7   10    1    8    9    2    4    3
    8    3    9    7   10    1    2    6    5    4
    7    1   10    8    2    9    3    4    6    5
    9    4    1    2    3   10    8    5    7    6
    3    6    5    1    4    2   10    9    8    7
    2    7    4    5    9    3    6   10    1    8
    5    8    2    3    6    4    1    7   10    9


Note: What we have called a ‘partial Latin square’ is
      referred to in the literature as a ‘complete
      Latin rectangle’.

File:      latin.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Mon Oct 12 03:55:17 EDT 2015

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

```
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 char-
acterized 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 represen-
tations 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 charac-
ters, 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

```
Reverse Draw
------- ----

Sparky, the proprietor of Sparky’s Fat and Fun Eatery
(she first named it ‘Slim and Slick’, but that did not
work out), has acquired a new sign making machine.  This
treats the paper-board as having a grid of squares.  The
machine has a head that can move over the paper board
and a pen that can be lowered to draw a black line, or
‘stroke’, while the head moves.

The machine has the following commands:

    pendown     Put pen down to start stroke.
    penup       Put pen up to end stroke.
    move D      Move in direction D one square.

where       D ::= N | NE | E | SE | S | SW | W | NW

meaning North, Northeast, East, ..., West, Northwest.
Here North is up, South is down, East is right, and
West is left.

The machine always starts and ends with the head in the
lower left corner and its pen up.  Its important that a
stroke be drawn all at once, by putting the pen down at
one end of the stroke and then moving along the stroke
to its other end before putting the pen up.  Putting the
pen down will draw a dot the size of a grid square at
the current position, and then moving to an adjacent
square with the pen down will not only draw a dot in the
adjacent square, but will also connect the two dots with
a dot-width line.

Sparky finds writing programs for this machine to be
quite a chore, and wants you to help.  She wants to
make signs that have line drawings, and not text, but
to control the machine she wants to input a representa-
tion of a sign as a text file like:
```

```
                                 CCC
                               CC
                             CC
                            C
                           C
                           C
                       AAAAAAAA
                   AAAA        AAAA
                 AA                AA
               AA                    AA
              A                        A
             A                          A
            A      BBB          EEE      A
            A     B   B        E   E     A
           A     B  F  B      E  G  E     A
           A     B  F  B      E  G  E     A
          A      B  F  B      E  G  E      A
          A       B   B        E   E       A
          A        B B          E E        A
          A                                A
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 M  K  H  H            D      D            R  Q  P  P
 M  K  H  H            D      D            R  Q  P  P
 M  K  H  H           D        D           R  Q  P  P
 M  K  H  H          D          D          R  Q  P  P
 M  K  H  H         D            D         R  Q  P  P
 M  K  H  H         D            D         R  Q  P  P
  MM KK HH          D            D          RR QQ PP
                     D          D
                      D        D
                       DDD  DDD
                          DD
```

which defines the ''strokes' that the machine is to make. Each stroke is labeled by a different capital letter. The machine repeatedly goes to one end of a stroke, executes a pendown, moves to the other end of the stroke, executes a penup, and then moves to one end of the next stroke, or if done with all strokes, moves to its starting position. The machine never omits the penup at the end of a stroke, even if the next stroke begins in an adjacent square.

Each stroke has just two ends. The continuation of a stroke involves going to one of the 8-neighbors of the previous square in the stroke. Each square in the stroke will have exactly 1 such neighbor if the square is a stroke end, and exactly 2 neighbors otherwise. There are NO strokes that have only one square, so the start and end of a stroke are different squares.

You are to take the text file and produce a program for the machine that has as few instructions as possible, and therefore executes as fast as possible.

However, this task has two parts. The first part determines the order in which the strokes will be visited, and which end of each stroke will be the stroke-drawing start point. The second part computes the instructions from the output of the first part. In this problem, you are only responsible for the first part of the task.

Lastly, and quite importantly, Sparky wants to have the computer already in the machine run your program. But this computer has only 128 megabytes of memory, of which some is taken by its operating software. Adding up the space taken by the operating software, you find there are 100 megabytes left over for your data. After considerable thought, you have agreed that this will suffice if you limit the input to a maximum of 20 strokes.

Input

For each of several test cases, a line containing just the test case name, followed by the input text followed by a line containing just '*'. The input text lines contain only single spaces and capital letters, and define strokes obeying the rules given above.

Input ends with an end of file.

No line is longer than 500 characters and there are no more than 500 input text lines (not counting the test case name and '*' lines) in a test case. Input text lines are not necessarily the same length; they may or may not be padded at the end with single spaces.

There are no more than 20 separate stokes in a test case (no more than 20 different capital letters appear in the input text lines).

Output

For each test case, first an exact copy of the input test case name line, then lines describing the strokes to be visited in the order they are to be visited, and then a line containing just '*'

The stroke describing lines have the form

```
        L: (x1,y1) ---> (x2,y2)
```

where L is the upper case letter labeling the stroke, (x1,y1) is the point at which drawing the stroke starts, and (x2,y2) is the point at which drawing the stroke stops.

The xy-coordinates are such that (0,0) is at the lower left of the input text, the place where the machine head starts and finishes with its pen up.

There is always more than one possible answer (in particular, making the machine trace its path in reverse order always works). Output only one answer.

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

```
-- SAMPLE 1 --
     A
    A A
   A   A
  A  B  A
   A B A
    ABA
     B
     B
   CCCCC
*
-- SAMPLE 2 --
[text for creature as above; or see sample.in]
*
```

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

```
-- SAMPLE 1 --
A: (4,3) ---> (6,3)
B: (5,5) ---> (5,1)
C: (7,0) ---> (3,0)
*
-- SAMPLE 2 --
M: (3,4) ---> (1,10)
X: (0,11) ---> (53,11)
P: (52,10) ---> (49,10)
Q: (46,10) ---> (48,4)
R: (45,4) ---> (43,10)
G: (33,17) ---> (33,15)
E: (34,13) ---> (32,13)
D: (30,10) ---> (23,10)
B: (21,13) ---> (19,13)
F: (20,15) ---> (20,17)
C: (27,26) ---> (35,31)
A: (43,12) ---> (10,12)
H: (10,10) ---> (7,10)
K: (4,10) ---> (6,4)
*
```

File: reversedraw.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Sat Oct 10 06:38:03 EDT 2015