```
PRACTICE PROBLEMS (Last Year's Contest Problems)Fri Apr  1 0
5:00:16 EDT 2016


Easier problems are first.

    piglatin
        High school keeps coming back.
        Boston Preliminary 2007

    turtledraw
        The Art of Turtle
        Boston Preliminary 2009

    convoy
        Traffic accordions.
        Boston Preliminary 2003.

    blowfish
        Scramble faster.
        Boston Preliminary 2006

    simplefsm
        Recognizing when you are synchronized.
        Boston Preliminary 2014

    pseudopi
        Throwing chalk.
        Boston Preliminary 2005.

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

```
Pig Latin
---------

You have been asked to translate English words to Pig
Latin.  The translation is very simple: take all the
consonants at the beginning of the word, move them to
the end, and add 'ay'.  If there are no consonants at
the beginning of the word, just add 'ay' to the end.
The consonants are all letters except 'a', 'e', 'i',
'o', 'u', and 'y'.   Note that 'y' is NOT a consonant
for our purposes.


Input
-----

A sequence of lines each containing an English word.
There are no spaces in any line.  Words will contain
only lower case letters.

The input ends with an end of file.

Output
------

For each English word, one line containing nothing but
the translation of the word into Pig Latin.

Example Input
------- -----

you
help
me
to
understand
pig
latin
this
hour


Example Output
------- ------

youay
elphay
emay
otay
understanday
igpay
atinlay
isthay
ourhay


Note: Actual Pig Latin moves only initial consonant
SOUNDS, and therefore does not move unsounded initial
consonants.  Thus 'hour' would become 'houray' in
actual Pig Latin.  There are also variants which put
'way' or 'yay' or some such at the end of words that
begin with a vowel sound.
```

File: piglatin.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Wed Oct 10 03:31:33 EDT 2007

Turtle Draw

A beaver, a dog, a frog, and a man were sharing a corner of a pond on a sweltering evening in early August. The beaver was considering air conditioning, the frog was imagining a water fall, the dog was happy to just swim after tennis balls, and the man knew he needed to write a computer program.

When he got home the man wrote a program called ‘turtle-draw’, in honor of the turtle that lived in the pond. She was not with the foursome that particular August evening, which just as well, as a 40 pound snapping turtle is a bit of a pond party pooper.

Input

The input contains a series of commands for an imaginary turtle living on an infinite board of squares. At any time, the turtle is on a particular square, and is facing in one of four directions, up, right, down, or left. In the beginning the turtle is facing up and all squares are blank.

The commands are:

```
        M               Move forward one square.
        L               Turn left 90 degrees.
        R               Turn right 90 degrees.
        <other>         Any other non-blank character:
                        write the character on the
                        current square and THEN move
                        forward one square.
```

The input is a sequence of test cases. Each test case begins with a line that names the test case. This is followed by one or more lines which contain commands for the turtle. No command line contains whitespace characters, and no command line contains just the character ‘.’. The test case ends with a line containing just ‘.’ (exactly one ‘.’).

In any test case the turtle will not wander more than 100 squares in any direction away from its starting position. No input line will contain more than 80 characters.

Output

For each test case, first one line that is an exact copy of the test case name line, then a single empty line (with no characters), and then just the portion of the infinite board that contains non-blank squares. Specifically, this portion of the board should NOT have any blank lines at its top or bottom, or any blank columns at its left or right edges. At the end of the test case, right after the portion of the board with no blank lines, there should be a single blank line.

Thus the output for each test case should have exactly two blank lines: the second line (after the name and before the board), and the last line (after the board). The entire output for ALL test cases ends with a blank line (if you get a ‘format error’ score you may have the blank lines wrong).

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

--SIGN--
EWRMGORMDOWNLMTHENLMPU
.
--HAT--
L/_M____M_\LL
MMML|RM____RM|
.
--DOG--
RR***LMR***LMR***L**RML****LMR**L****RML***RML***
L//////_\\\\\\L\**RML****
LLMRMMMMMMMMMMMMMMMMML****LMR**
LMMMMMMMMMMLMMMML--MMM--
.
```

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

--SIGN--

 GO
W  D
E  O
   W    U
   N    P
    THEN

--HAT--

   ____
\_|____|_/

--DOG--

 \\\\\\_//////
 **            **
 **            **
* *            * *
*  * --   -- *  *
*  *         *  *
*  *         *  *
    *       *
    *       *
    *       *
    **    ***
      ****

NOTE: This output ends with a single blank line.
```

```
File:      turtledraw.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Thu Oct 15 05:42:30 EDT 2009
```

Convoy

The Safe and Speedy Driver Company makes robot drivers. They have been asked to provide drivers for convoys of trucks, but are unsure if their basic traveling algorithm will work. You have been asked to simulate it, so see under what circumstances it will cause crashes.

The simulation is of N trucks traveling from right to left. In the beginning, all trucks are separated by exactly L0 feet and are traveling a velocity V0 ft/s. The simulation lasts for some number of seconds.

Each driver decides at the beginning of each second whether to accelerate during the second, decelerate (brake) during the second, or maintain velocity during the second. All acceleration is by A0 ft/s/s. All deceleration is by -A0 ft/s/s unless the current truck velocity V ft/s is less than A0, in which case the deceleration is by -V ft/s/s so the truck velocity will be 0 at the end of the second. Maintaining velocity, of course, involves an acceleration of 0 ft/s/s during the second.

During each second, the acceleration in ft/s/s of each truck is constant, the velocity of the truck is a linear function of time, and the distance traveled by the truck is a quadratic function of time.

The algorithm each driver of a non-lead truck follows to determine the truck’s acceleration at the beginning of a second depends upon parameters dV, dL, A0, and L0, and is as follows:

    if the truck is approaching the truck it is following at a relative velocity of at least dV ft/s, then decelerate

    else if the truck is receding from the truck it is following at a relative velocity of at least dV ft/s, then accelerate

    else if the truck is at least L0+dL ft from the truck it is following, accelerate

    else if the truck is at at most L0-dL ft from the truck it is following, decelerate

    else maintain speed

The lead truck receives instructions that tell its driver what to do for each second.

```
Input
-----

For each of several test cases,

    One line containing just the test case name.

    One line containing the numbers

        N L0 dL V0 dV A0

    One line containing the instructions for the lead
    driver.

The instructions are a sequence of +, 0, and - charac-
ters, one character per second.  Each is interpreted as
an instruction for the lead driver to

    +   accelerate
    0   maintain speed
    -   decelerate

for one second.  The leftmost character is for the first
second, the rightmost for the last second, and the num-
ber of instruction characters is the number of seconds
in the simulation.  There are no spaces in the instruc-
tions line.

Simulations are limited to at most 100 seconds and at
most 11 trucks.  Input ends with an end of file.

Output
------

For each case

    One line containing an exact copy of the test case
    name line.

    Lines containing the distances between the non-lead
    trucks and the truck they are following.  Each line
    contains N-1 distances, each in exactly 8 columns
    with exactly 3 decimal places.  Each distance is the
    distance between a truck and the truck it is follow-
    ing.  The distances are for the trucks from left to
    right: the first is for the truck after the lead
    truck.

    One line with the initial distances is printed,
    followed by one line for each second of simulation
    with the distances at the end of the second.

    If any output line has a negative distance, the
    simulation terminates, and a next line containing
    just 'CRASH' is output.

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

-- SAMPLE 1 --
3 88 11 44 11 44
-----
-- SAMPLE 2 --
3 88 11 44 11 44
+++++0000--------------
-- SAMPLE 3 --
5 88 11 44 11 44
+-----
```

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

-- SAMPLE 1 --
  88.000  88.000
  66.000  88.000
  44.000  66.000
  44.000  44.000
  44.000  44.000
  44.000  44.000
-- SAMPLE 2 --
  88.000  88.000
 110.000  88.000
 154.000 110.000
 198.000 154.000
 242.000 198.000
 286.000 242.000
 308.000 286.000
 286.000 330.000
 264.000 330.000
 242.000 330.000
 198.000 330.000
 154.000 286.000
 110.000 242.000
  66.000 198.000
  22.000 154.000
 -22.000 110.000
CRASH
-- SAMPLE 3 --
  88.000  88.000  88.000  88.000
 110.000  88.000  88.000  88.000
 110.000 110.000  88.000  88.000
  66.000 110.000 110.000  88.000
  44.000  66.000 110.000 110.000
  44.000  44.000  66.000 110.000
  44.000  44.000  44.000  66.000
```

```
Postscript
----------

Driving safety experts recommend drivers maintain a
at least 3 seconds separation between themselves and
the car in front of them in good weather.



File:      convoy.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sat Feb 21 02:52:12 EST 2015
```

```
Mini-Blowfish
---- --------

The Blowfish algorithm has become a popular encryption
algorithm for data streams and large files, as it can
be efficiently implemented in software.  In this problem
you are asked to code and test a miniature version of
this algorithm, which we call Mini-Blowfish, or MB for
short.

Description of MB:
----------- -- --

MB uses an 18+256 byte vector of ‘subkeys’.   The first
18 of these are referred to as P[1] through P[18].  The
next 256 are referred to as S[0] through S[255].  The
18+256 subkeys are collectively referred to as K[1]
through K[18+256], so K[1] == P[1], K[18] == P[18],
K[19] == S[0], K[18+256] == S[255].

The S values define a ‘substitution-box’, or S-box, that
takes a byte B as input and returns the byte S[B] as
output, where bytes are viewed as unsigned integers from
0 through 255.  E.g., if B == 5 the S-box returns S[5].

The data encryption algorithm inputs and outputs 16 bit
blocks.  These are divided into a high order byte, HB,
and a low order byte LB, so block B == 256 * HB + LB.

The encryption algorithm is:

    Input B = 256 * HB + LB.
    For round R = 1 though 16:
        HB = HB xor P[R].
        LB = LB xor S[HB].
        swap HB and LB.
    Finishing:
        swap HB and LB (undo the round 16 swap).
        LB = LB xor P[17];
        HB = HB xor P[18];
    Output B = 256 * HB + LB.

Note that P[1], ..., P[18] are accessed in order by the
encryption algorithm.  Decryption uses the same algori-
thm except that P[1], ..., P[18] are used in the reverse
order (P[18] is used in round 1 and P[1] is xor’ed at
the end into HB).

The main idea in Blowfish is the method of computing the
subkeys.  In fact, the idea is to have a lot of subkeys
(full Blowfish as 1042 32-bit subkeys).  Computing the
subkeys takes a long time, so changing the key in MB or
Blowfish is slow, and has been made so in order to have
a secure algorithm in which encrypting the data given
the subkeys is fast.

To initialize the subkey vector K[1], ..., K[18+256] you
need as input a password, which is any string of
characters.  Let the bytes of the password be W[1],
W[2], ..., W[N] where N is the length of the password.
The MB subkey computation algorithm is then:
```

```
        Input W[1], ..., W[N].
        For i from 1 through 18+256:
            K[i] = 7 ** i mod 256;
        For i from 1 through N:
            K[i] = K[i] xor W[i];
        Set B = 0, a 16 bit value.
        For round Q from 1 through (18 + 256)/2:
            Encrypt B to obtain Encrypted-B
            Set B = Encrypted-B
            Let B = 256 * HB + LB as above.
            Set K[2*Q-1] = HB and K[2*Q] = LB.
        Output K[1], ..., K[18+256].
```

Note that the output B of the encryption in round Q becomes the input B to the encryption in round Q+1. Also the subkeys at the end of round Q are the subkeys used in the encryption in round Q+1. Thus B and the subkeys keep changing as Q advances. The subkeys at the end of round Q = (18+256)/2 are the final output of the subkey computation algorithm.

Input

Lines each of which contains a password and some integers to be encrypted using the password, all followed by the integer -1 (which is NOT to be encrypted). These are separated from each other by whitespace. No line is longer than 80 characters.

The password is a string of one or more letters and digits, each interpreted as a byte equal to the ASCII code of the letter or digit (ASCII codes are the codes used to represent characters as integers in modern computers, and all ASCII codes are between 0 and 127). The integers to be encrypted are all in the range from 0 through 65535 (= 2**16 - 1), and each integer represents a 16 bit block.

Input ends with an end of file.

Output

For each input line, one output line, in the same format as the input line, except that each integer to be encrypted is replaced by the result of encrypting it.

Sample Input

```
abcdefg 0 1 2 3 4 5 -1
2hotfudge 28647 64826 42873 60872 53872 7648 29640 -1
```

Sample Output

```
abcdefg 61669 41297 34644 22212 18368 679 -1
2hotfudge 37515 44577 40580 64732 42141 33306 62416 -1
```

```
Further Information
------- -----------

Blowfish was invented by Bruce Schneier, and is describ-
ed at
        www.schneier.com/paper-blowfish-fse.html

Blowfish is one of a large number of ‘Feistel ciphers’.
The full algorithm uses 32 bit subkeys, 64 bit blocks,
and 4 S-boxes that are applied to the 4 bytes of a
32-bit half-block to get 4 32-bit half-blocks that are
combined using addition and exclusive-or to make one
32-bit half-block.  The subkeys are initially set to
the fractional digits of PI, which are assumed to be
random.  Short passwords are extended by cycling through
their bytes.  However, like MB full Blowfish also has 16
rounds, 18 P values, and the same control flow as MB.


File:      blowfish.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct 10 02:26:24 EDT 2006
```

```
Simple FSM
------ ---

Your company wants to implement a very simple Finite
State Machine (FSM) to recognize certain patterns in an
input bit stream.  For example, they want to know when
the last 7 bits are '1111110'.  This bit sequence is
used in a 'flag' that separates data blocks, where the
data has been modified so that the flag bit sequence can
never occur within a data block.

You have been asked to write a basic FSM simulator.
Input is a string of bits.  States are labeled with
upper case letters or the special characters '$' and
'*'.

A simple example FSM description is:

        $ $ A
        A * A
        * $ A

which describes an FSM that is in state * when the last
2 bits read are '10'.

The 3 lines of this FSM description say:

    when in state $, on reading a 0 go to state $,
                 but on reading a 1 go to state A

    when in state A, on reading a 0 go to state *,
                 but on reading a 1 go to state A

    when in state *, on reading a 0 go to state $,
                 but on reading a 1 go to state A

For each state you are given two successor states, one
to go to if the next bit input is '0', and one to go to
if the next bit input is '1'.  The FSM starts in state
'$', and stops when there are no more bits to read.

If the FSM does what it is supposed to, it will be in
state '*' if and only if the last several bits read are
the pattern sought.

To see how this FSM executes when inputting the binary
string '010011001110011110', write the sequence of
states that the machine is in so that each state is
underneath the next bit to be read:

        0100110011100111100
        $$A*$AA*$AAA*$AAAA*$

This means that the FSM:

    starts in state '$'
    goes to state '$' upon reading the first '0'
    goes to state 'A' upon reading the first '1'
    goes to state '*' upon reading the second '0'
    goes to state '$' upon reading the third '0'
    . . . . . . . . . . .
    goes to state 'A' upon reading the last '1'
    goes to state '*' upon reading the next to last '0'
    goes to state '$' upon reading the last '0'
    stops when there is no binary digit left to read

Similarly the FSM:

        $ $ A
        A $ B
        B $ C
        C * C
        * $ A

which is intended to recognize the pattern '1110'
executes as follows on the same input string:

        0100110011100111100
        $$A$$AB$$ABC*$ABCC*$
```

```
Input
-----

For each of several test cases, the following in order:

    a line containing just the test case name
    lines containing the FSM description
    a line containing just ‘.’
    one or more input lines each containing
        a binary string
    a line containing just ‘.’

No line is longer than 80 characters.

Input ends with an end of file.

An FSM description consists of ‘state description lines’
each of the form:

        s z n

which says:

    when in state s, on reading a 0 go to state z,
                 but on reading a 1 go to state n

There are 28 FSM states ($, A, B, ..., X, Y, Z, *), but
unused states have no state description line.

Each binary string contains just ‘0’s and ‘1’s and is
processed independently of the other binary strings.
```

```
Output
------

For each test case, first an exact copy of the test
case name line, and then for each binary string input
line two lines:

    (1) an exact copy of the binary string input line
    (2) the state sequence of the FSM execution for
        the input binary string, as described above;
        here the state that the FSM is in just before
        reading a binary digit is placed directly under
        the binary digit

Note there is no whitespace in either of these two
lines.
```

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

-- RECOGNIZE '10' --
$ $ A
A * A
* $ A
.
0000
1111
0100
010010100
0100110011100111100
.
-- RECOGNIZE '1110' --
$ $ A
A $ B
B $ C
C * C
* $ A
.
1111
1100
1110
1011
101010
010010100
0100110011100111100
0101101110111100
.
```

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

-- RECOGNIZE '10' --
0000
$$$$$
1111
$AAAA
0100
$$A*$
010010100
$$A*$A*A*$
0100110011100111100
$$A*$AA*$AAA*$AAAA*$
-- RECOGNIZE '1110' --
1111
$ABCC
1100
$AB$$
1110
$ABC*
1011
$A$AB
101010
$A$A$A$
010010100
$$A$$A$A$$
0100110011100111100
$$A$$AB$$ABC*$ABCC*$
0101101110111100
$$A$AB$ABC*ABCC*$
```

```
Extra Notes (not relevant to solving problem):
----- -----  --- -------- -- ------- -------

The synchronous High Level Data Link Control (HDLC)
protocol for communications via synchronized bit
streams uses ‘01111110’ as a ‘flag’.  Successive flags
are used to synchronize clocks and bracket data blocks,
which are altered so they cannot contain flags.  This is
done simply by inserting a 0 bit whenever 5 successive 1
bits have occurred in the data, and removing the 0 bit
when ‘111110’ is received in data.  The flag contains 6
successive 1’s; 7 successive 1’s is considered to be a
transmission error.

All FSM’s have the same structure except for input/out-
put.  As an example of a slightly more complicated in-
put/output structure, let the FSM output a character
when making a transition from one state to another.
So the description line ‘s z n’ becomes ‘s z/Z n/N’
where Z and N are characters output when the next
state becomes z or n, and the output is optional, so
the ‘/Z’ and ‘/N’ are optional.  Using this you can
describe an FSM that will insert or remove the 0 bit
in HDLC data.

A more substantial modification replaces the input
string with a tape that can move backward or forward
one position, so instead of outputting a character the
machine writes a new digit at the current position
and then moves the tape either backward or forward one
position.  If we also replace the set {0,1} of two
digits by an arbitrary finite set of ‘symbols’, we have
what is called a ‘Turing Machine’.
```


```
File:       simplefsm.txt
Author:     Bob Walton <walton@seas.harvard.edu>
Date:       Tue Oct  7 00:10:16 EDT 2014
```

```
Pseudo-Random Computation of PI
-------------------------------

One of the classic demonstrations of probability is the
following.  The professor draws a large square on the
blackboard, and draws its inscribed circle.  Then stand-
ing with her back to the board, she throws pieces of
chalk at the square.  After this she counts the number M
of hits in the circle and the number N >= M of hits in
the square (including those in the circle), and demon-
strates that M/N is about PI/4.  This is because PI/4 is
the area of the inscribed circle divided by the area of
the square, and the probability of hitting any small
part of the square is roughly identical to hitting any
other small part of the square.

You have been asked to simulate the demonstration in the
computer.  The square is to be simulated by the unit
square in the XY-plane, [0,1]x[0,1], which has (0,0) as
its lower left corner and (1,1) as its upper right
corner.  To simulate throwing the chalk, two random
integers X and Y are ‘drawn uniformly’ (see below for
details) from the range 0 .. S-1, where S > 0 is some
integer.  Then the coordinates where the chalk strikes
are set at ((X + 0.5)/S, (Y + 0.5)/S ).  These are
inside the square, so all our ‘throws’ count toward N.
They are inside the circle, and count toward M, if and
only if the chalk strikes at a distance of 0.5 or less
from the center of the circle, (0.5, 0.5).

Thus if S = 100 and the first two random integers drawn
are 37 and 69, the chalk point is (0.375,0.695) which
is distance 0.23 from (0.5,0.5), and is therefore in the
circle and counts toward both M and N.


Drawing Random Numbers
------- ------ -------

You are asked to draw pseudo-random numbers according to
the equation:

        RANDOM = ( RANDOM * MULTIPLIER ) mod MODULUS

where RANDOM is the value of the pseudo-random number,
the equation steps from the the last pseudo-random
number to the next pseudo-random number, and MULTIPLIER
and MODULUS are fixed values that determine the pseudo-
random number sequence.

To get started, RANDOM is initialized to a value called
SEED.  The first pseudo-random number in the sequence
is not SEED, but the first number after SEED in the
sequence.

If MULTIPLIER and MODULUS have good values for this
purpose, the resulting sequence of numbers appears when
tested to be truly random and uniformly distributed in
the range from 1 through MODULUS - 1.  Uniformly dis-
tributed means all values in this range are equally
probable.  The choices

        MULTIPLIER = 7**5 = 16807
        MODULUS = 2**31 - 1 = 2147483647

are very good for this purpose.

For example, if MULTIPLIER and MODULUS are as just
given, and the SEED is 374332679, then the first two
random numbers are 1429733890 and 1342962947.

A remaining difficulty is how to convert uniformly
distributed integers from 1 through MODULUS - 1 to
uniformly distributed integers from 0 through S-1.  An
easy solution, which we will adopt, is to set
```

```
        S = MODULUS - 1

and subtract 1 from each value of RANDOM.  Thus ‘a chalk
throw’ is simulated by executing

    RANDOM = ( MULTIPLIER * RANDOM ) mod MODULUS
    X = RANDOM - 1
    X = (X + 0.5 ) / S
    RANDOM = ( MULTIPLIER * RANDOM ) mod MODULUS
    Y = RANDOM - 1
    Y = (Y + 0.5 ) / S

to yield (X,Y) in the unit square.

Implementation of the above algorithm requires integers
longer than 32 bits.  In C or C++ you can use doubles
and the fmod function.  Or you can use ‘long long’s and
the % operator.  In JAVA you can use ‘long’s and the %
operator.  Remember, ‘long’s are only 32 bits in C and
C++, but are 64 bits in JAVA.  ‘long long’s are 64 bits
in C and C++.

Input
-----

For each of several test cases, one line containing
four numbers in the order:

        N MULTIPLIER MODULUS SEED

The numbers may be separated by spaces or tabs.  All
input numbers are positive integers below 2**31 (but
some products computed by intermediate computations will
be larger).

Input ends with an end of file.

The simulation is to be done with RANDOM initialized to
SEED (SEED is NOT the first pseudo-random number) and
S = MODULUS - 1.


Output
------

For each test case one line containing five numbers in
the order:

        N MULTIPLIER MODULUS SEED PI_ESTIMATE

where the first four numbers are copied from the input,
and PI_ESTIMATE equals 4*M/N expressed as a decimal
number with exactly 5 decimal places.

Example Input
------- -----

100      16807 2147483647 374332679
1000     16807 2147483647 374332679
10000    16807 2147483647 374332679
100000   16807 2147483647 374332679
1000000  16807 2147483647 374332679


Example Output
------- ------

100 16807 2147483647 374332679 3.20000
1000 16807 2147483647 374332679 3.13600
10000 16807 2147483647 374332679 3.15960
100000 16807 2147483647 374332679 3.13888
1000000 16807 2147483647 374332679 3.14167
```

File: pseudopi.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Wed Oct 19 07:19:20 EDT 2005

```
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