The task is to develop a program that places points in a 2D plane such that the points form parallelograms.

There are given three points A, B, and C in a 2D plane. The points are described by their coordinates (pairs of decimal numbers). Your program reads the coordinates from the standard input. Next, the program computes the coordinate of the fourth point such that the points form a parallelogram (or a special variant of a parallelogram: a rhombus, a rectangle, or a square). The coordinate of the fourth point is displayed in the output, next, the program displays the name of the shape (square/rectangle/rhombus/parallelogram). In general, there are 3 possible ways to place the fourth point: parallelograms ABA'C, ABCB', and AC'BC. The program shall display all three possible results, see the sample runs below.

The problem does not have any solution if the three input points share a common line. Your program must detect this special case and output accordingly (see sample runs below).

The program must validate input data. If the input is invalid, the program must detect it, it shall output an error message (see below) and terminate. If displayed, the error message must be displayed to the standard output (do not send it to the error output) and the error message must be terminated by a newline (\n). The input is considered invalid, if:

- the coordinates are invalid (are not valid decimals),
- some coordinate(s) are missing,
- there are missing or there are some extra separator characters (each coordinate is given as two decimals, the decimals are enclosed with square brackets and separated by a comma).

Sample program runs:

```
Point A:
[0, 0]
Point B:
[7, 0]
Point C:
[3, 2]
A': [10,2], parallelogram
B': [-4,2], parallelogram
C': [4,-2], parallelogram
```

```
Point A:
[0,0]
Point B:
  [  5  ,  0     ]
Point C:


[3,
4
]
```

A': [8,4], rhombus
B': [-2,4], parallelogram
C': [2,-4], parallelogram

Point A:
[0,0]
Point B:
[-3,4]
Point C:
[4,3]
A': [1,7], square
B': [7,-1], parallelogram
C': [-7,1], parallelogram

Point A:
[10.5, 10.5]   [12.5, 10.5][10.5, 15e+0]
Point B:
Point C:
A': [12.5,15], rectangle
B': [8.5,15], parallelogram
C': [12.5,6], parallelogram

Point A:
[0, 0]
Point B:
[3, 3]
Point C:
[10, 10]
Parallelograms do not exist.

Point A:
[0, 0]
Point B:
[2270.242, 0]
Point C:
[234.08, 2258.142]
A': [2504.322,2258.142], rhombus
B': [-2036.162,2258.142], parallelogram
C': [2036.162,-2258.142], parallelogram

Point A:
[740.865, 887.560]
Point B:
[340.090, 1241.872]
Point C:
[1095.177, 1288.335]
A': [694.402,1642.647], square
B': [1495.952,934.023], parallelogram
C': [-14.222,841.097], parallelogram

Point A:
[-306.710, -894.018]
Point B:
[6369.015, 66159.129]
Point C:
[6016.590, 62619.258]
Parallelograms do not exist.

Point A:
[2, 5]
Point B:
[3, abcd]
Invalid input.

Point A:
[2, 5]
Point B:
[3, 4]
Point C:
[7 9]
Invalid input.

Advice:

- The sample runs above list both the output of your program (bold face text) and user input (regular text). The bold/regular formatting is included here, in the problem statement page, to increase readability of the listing. Your program must output the text without any additional markup.
- Do not forget the newline (\n) after the last output line.
- Use `double` data type to represent the values. Do not use `float`, the precision of `float` is not always sufficient.
- The program can be developed without additional functions (i.e. in one big `main`). However, if divided into functions, the program is readable and easier to debug.
- All numeric values in the input fit into the range of `double` data type. The reference uses `double` and `int` data types to represent numbers.
- Input data can be read by means of `scanf` function. Have a look at `"%c"` and `" %c"` conversions (without and with a space before the conversion, read manual to learn the difference).
- Please strictly adhere to the format of the output. The output must exactly match the output of the reference program. The comparison is done by a machine, the machine requires an exact match. If your program provides output different from the reference, the program is considered malfunctioning. Be very careful, the machine is sensitive event to whitespace characters

(spaces, newlines, tabulators). Please note that all output lines are followed by a newline character (\n). This applies even to the last line of the output, moreover, this applies even to the error message. Download the enclosed archive. The archive contains a set of testing inputs and the expected outputs. Read Progtest FAQ to learn how to use input/output redirection and how to simplify testing of your programs.

- The the coordinates in your output are compared to the coordinates in the reference output. The comparison accepts small differences up to 1 ‰.
- Your program will be tested in a restricted environment. The testing environment limits running time and available memory. The exact time and memory limits are shown in the reference solution testing log. However, neither time nor memory limit could cause a problem in this simple program. Next, the testing environment prohibits the use of functions which are considered "dangerous" (functions to execute other processes, functions to access the network, ...). If your program uses such functions, the testing environment refuses to execute the program. Your program may use something like the code below:

```
int main ( int argc, char * argv [] )
{

  ...

  system ( "pause" ); /* prevent program window from closing */
  return 0;
}
```

This will not work properly in the testing environment - it is prohibited to execute other programs. (Even if the function were allowed, this would not work properly. The program would infinitely wait for a key to be pressed, however, no one will press any key in the automated testing environment. Thus, the program would be terminated on exceeded time limit.) If you want to keep the pause for your debugging and you want the program to be accepted by the Progtest, use the following trick:

```
int main ( int argc, char * argv [] )
{

  ...

#ifndef __PROGTEST__
  system ( "pause" ); /* this is ignored by Progtest */
#endif /* __PROGTEST__ */
  return 0;
}
```

- Textual description of valid input data structure is not 100% exact. Therefore we provide a formal specification of the input language in EBNF:

```
    input      ::= { whiteSpace } coord { whiteSpace } coord { whiteSpace }
coord { whiteSpace }
    whiteSpace ::= ' ' | '\t' | '\n' | '\r'
```

```
    coord      ::= '[' { whiteSpace } decimal { whiteSpace } ',' { whiteSpace
} decimal { whiteSpace } ']'
    decimal    ::= [ '+' | '-' ] integer [ '.' integer [ ( 'e' | 'E' ) [ '+'
| '-' ] integer ] ]  |
                   [ '+' | '-'  ] '.' integer [ ( 'e' | 'E' ) [ '+' | '-' ]
integer ]
    integer    ::= digit { digit }
    digit      ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```