Triangles #2

Submission deadline:	**2018-11-11 23:59:59**
Late submission with malus:	**2019-01-06 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**5.5000**
Max. assessment:	**5.0000** (Without bonus points)
Submissions:	14 / 20 Free retries + 10 Penalized retries (-10 % penalty each retry)
Advices:	2 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to develop a program that compares two given triangles in a 2D plane. The problem is an extension of the simple variant. This version accepts input triangles entered either by their vertices or by the lengths of their sides. Starting with the simple problem is highly recommended.

There are given two triangles in a 2D plane. Each triangle is defined either by its vertices or by the lengths of its sides:

- triangle defined by its vertices is entered in the form of three coordinates. The three coordinates are enclosed in curly braces; individual coordinates are separated by commas. A coordinate consists of two decimal numbers separated by a semicolon, the two numbers are enclosed in square braces. An example:

  ```
  { [ 1.5; 2 ], [3;4.2], [ 0.5 ; 0.6 ] }
  ```

- triangle defined by its sides is entered in the form of three decimal numbers. The three numbers are enclosed in curly braces and separated by commas. An example:

  ```
  {2.1, 3, 3.4}
  ```

Your program reads the triangles from the standard input. Next, the program decides one of the 4 cases:

- the input does not form a triangle (the three vertices defining the triangle are on a common line, or the sides do not obey triangle inequality),
- the triangles are identical,
- the triangles are not identical while their circumference is the same, or
- the triangles are entirely different.

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 sent 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 or sides are invalid (are not valid decimals),
- some coordinate(s) or sides are missing,
- there are missing some separators (braces, commas, semicolons).

Sample program runs:

```
Triangle #1:
{[0;0],[5;0],[2.5;3]}
Triangle #2:
 { [ 4 ; -1 ] , [ 7 ; 1.5 ] , [ 4 ; 4 ] }
The triangles are identical.
```

```
Triangle #1:
{
[
0
;
15
]
,       [       112     ;       0       ]       ,[112;15]}
Triangle #2:
{96.0,40.0,104.0}
The triangles are not identical, however, they have the same circumference.
```

```
Triangle #1:
 { 10 , 15 , 10 }
Triangle #2:
{12,8,17}
Triangle #2 has a longer circumference.
```

```
Triangle #1:
{[0;14.04],[11.2;0],[0;0]}
Triangle #2:
{[20.16;0],[0;2.7],[20.16;2.7]}
The triangles are not identical, however, they have the same circumference.
```

Triangle #1:
{[0.00;0.00],[0.24;0.70],[0.24;0.00]}
Triangle #2:
{0.65,0.78,0.25}
The triangles are not identical, however, they have the same circumference.

Triangle #1:
{51.09,49.49,4.94}
Triangle #2:
{37.92,50.11,17.49}
The triangles are not identical, however, they have the same circumference.

Triangle #1:
{[0;0],[10;0],[0;10]}
Triangle #2:
{[0;0],[10;10],[15;15]}
Invalid triangle.

Triangle #1:
{[10;0],[20;1],[25;1.5]}
Invalid triangle.

Triangle #1:
{1,2,3}
Invalid triangle.

Triangle #1:
{[0;0],[999.990;204.330],[899.991;183.897]}
Invalid triangle.

Triangle #1:
{1.923,59.240,61.163}
Invalid triangle.

Triangle #1:
{[1;2],[3;abcd],[7,9]}
Invalid input.

Advice:

- The sample runs above list both the output of your program (boldface font) and user input (regular font). 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.
- The input may be processed using the `scanf` function.
- `scanf` conversions `%c` and `%c` (with an extra space before %c) are quite useful in this assignment. Learn the difference in the manual pages.
- Please strictly adhere to the format of the output. The output must exactly match the output of the reference program. The comparison is completed by a machine, meaning an exact match is required. If your program provides output different from the reference, the program is considered malfunctioning. Be very careful since the comparison is sensitive even to whitespace characters (spaces, newlines, tabulators). Please note that all output lines are followed by a newline character (\n). The new line character must be present after the last line of the output and after the error messages. 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.

- 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 } triang { whiteSpace } triang { whiteSpace }
triang     ::= '{' (byCoord | bySides) '}'
byCoord    ::= {whiteSpace} '[' coord ']' {whiteSpace} ','
               {whiteSpace} '[' coord ']' {whiteSpace} ','
               {whiteSpace} '[' coord ']' {whiteSpace}
bySides    ::= {whiteSpace} decimal {whiteSpace} ','
               {whiteSpace} decimal {whiteSpace} ','
               {whiteSpace} decimal {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'
```

Sample data: Download

Reference

14 **2018-11-03 23:50:56** Download

Submission status: Evaluated

Evaluation: 5.5000

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Max. run time: 0.006 s (limit: 1.000 s)
    - Total run time: 0.063 s
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test': success
    - result: 100.00 %, required: 50.00 %
    - Max. run time: 0.004 s (limit: 1.000 s)
    - Total run time: 0.141 s
    - Optional test success, evaluation: 100.00 %
  - Test 'Invalid input test': success
    - result: 100.00 %, required: 50.00 %
    - Max. run time: 0.007 s (limit: 1.000 s)
    - Total run time: 0.520 s
    - Optional test success, evaluation: 100.00 %
  - Test 'Random test': success
    - result: 100.00 %, required: 50.00 %
    - Max. run time: 0.006 s (limit: 1.000 s)
    - Total run time: 0.098 s
    - Optional test success, evaluation: 100.00 %
  - Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00 * 1.00)
- Advices used: 2
- Penalty due to advices: None (2 <= 2 limit)
- Total percent: 100.00 %
- Early submission bonus: 0.50
- Total points: 1.00 * (5.00 + 0.50) = 5.50

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**2**	--	--	--
Lines of code:	**90**	**45.00 ± 37.00**	82	main
Cyclomatic complexity:	**37**	**18.50 ± 16.50**	35	main