Middle East Technical University
Department of Computer Engineering

CENG 111

Fall 2014

Take Home Exam 2
[**Version: 1.1**]

REGULATIONS

Due date: 12 December 2014, 23:59, Friday *(Not subject to postpone)*

Submission: Electronically. You should put your solution code into a file called `the2.py` and submit it through the COW web system. Resubmission is allowed (till the last moment of the due date); the last one replaces the previous ones.

Team: There is **no** teaming up. The take home exam has to be done/turned in individually.

Cheating: **This is an exam:** All involved parties (source(s) and receiver(s)) get zero, and will be subject to disciplinary action.

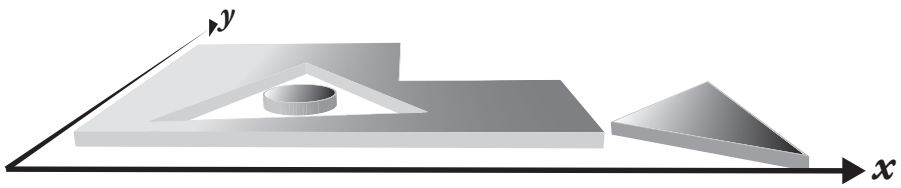
PROBLEM

This THE is about calculating the *Center of Mass* (abbreviated "CM" throughout this text) of a set of primitive geometric shapes. You will be given a sequence of geometric shapes, each of which is about the location of

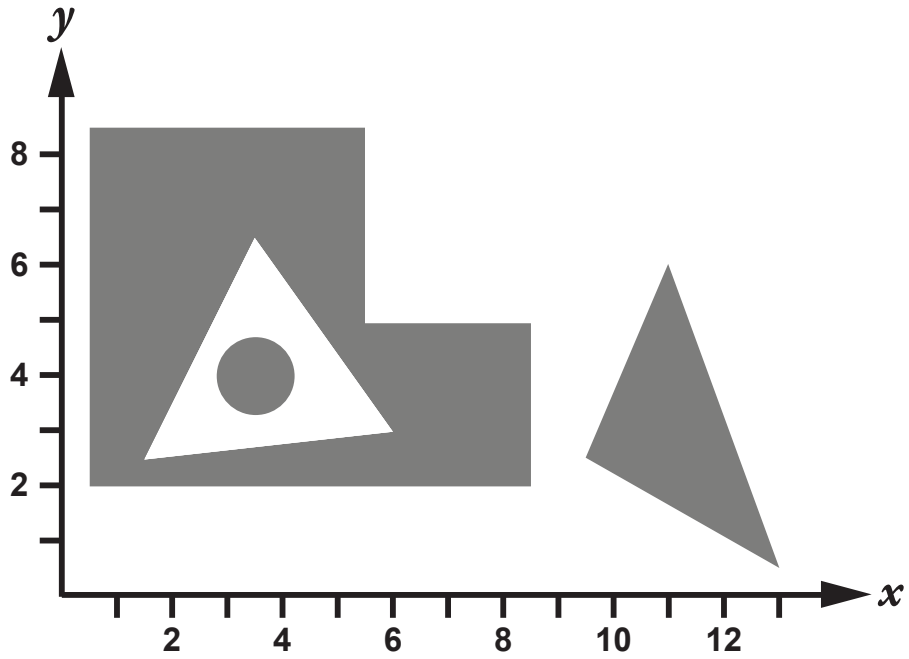
- a triangle, or
- a rectangle, or
- a circle

in the x-y plane. Each geometric shape either has a uniform distributed mass or is a hole within other geometric shapes. If it is a hole, it is guaranteed that it entirely resides in mass geometric shapes. In other words, every hole point removes a mass point. It is quite possible that a massy geometric shape has more than one hole in it. It is also possible that a hole can contain other massy geometric shapes. It is guaranteed, however, that, at no point of the x-y plane, two massy geometric shapes coexist (i.e., a double layer of massy objects is not possible).

Below you see an example:



A bird's-eye view of the same example is provided below:



The corresponding input that your function will receive is:

```
[ [1, 13.0, 0.5, 11.0, 6.0, 9.5, 2.5],
   [-1, 6.0, 3.0, 1.5, 2.5, 3.5, 6.5],
   [1, 0.5, 2.0, 8.5, 2.0, 8.5, 8.5, 0.5, 8.5],
   [1, 3.5, 4.0, 0.7],
   [-1, 8.5, 8.5, 5.5, 5.0, 5.5, 8.5, 8.5, 5.0]
]
```

The first item is the massy triangle; the second one the triangle hole; the third the massy rectangle; the fourth the massy circle, and finally, the fifth one is the rectangle hole. The order of items could be any possible permutation. Furthermore, the corners of a shape can be given in any possible order (any traversal order is possible).

SPECIFICATIONS

- You should code your solution into a `the2.py` file and your solution should be accessible via a function named `cm` that gets a single list argument.
- Each geometric shape data (massy or hole) is given as a sublist of the argument. As mentioned above, there is no order specified for the shapes.
- The argument of the function is composed of a set of sublists, which start with either `1` or `-1`. `1` signifies a mass shape, whereas `-1` does a hole.
- For a triangle or rectangle, the rest of the list contains (x, y) coordinates of the corners.

  For a circle, the rest of the list contains the (x, y) coordinates of the center followed by the radius.

  All coordinates and the radius data are floating point numbers.
- Mass of shapes is uniform and has $density = 1$(Unit mass)/(Unit area).

- Your function should return a list with three floating point numbers:
  [⟨x-coordinate of the CM⟩, ⟨y-coordinate of the CM⟩, ⟨total mass⟩]
  For the example given above, the output should be (see the note on precision loss):
  `[5.2145198383188527, 4.6759259766052423, 42.164380400258999]`

- For any point **p** in the x-y plane, the following holds:
  If the number of the mass shapes that contain **p** is m, and if the number of the hole shapes that contain **p** is h, then: $0 \leq m - h \leq 1$.

- There is no information about the maximum number of geometric shapes. **You should not attempt to store objects for later processing**. Rather, you are expected to get a sub-list from the input, do the relevant math that computes the contribution of that geometric object to the CM, and move on to the next shape. If you are in the need to return to a previous geometric object, you are definitely missing the correct algorithm.

- None of the geometric shapes are degenerate (i.e., having its area equal to zero).

- Take π as 3.1415926535897932.

- You can only use recursion, and only the built-in facilities of Python. You should not use functions from libraries. Your function should only return a value; it should not print anything on the screen.

EXAMPLE RUN

```
>>> L = [ [1, 13.0, 0.5, 11.0, 6.0, 9.5, 2.5],
   [-1, 6.0, 3.0, 1.5, 2.5, 3.5, 6.5],
   [1, 0.5, 2.0, 8.5, 2.0, 8.5, 8.5, 0.5, 8.5],
   [1, 3.5, 4.0, 0.7],
   [-1, 8.5, 8.5, 5.5, 5.0, 5.5, 8.5, 8.5, 5.0]
]
>>> cm(L)
[5.2145198383188527, 4.6759259766052423, 42.164380400258999]
```

EVALUATION

- There will be no erroneous test input.

- All submissions will be tested under strictly equal conditions on multiple data sets on Linux-running lab machines using Python version 2.7.6.

- Evaluation will take into account the possible errors due to truncation in floating point representation. You assume that two real numbers A and B are equal if $abs(A - B) < 0.000001$.