Imperial College
London

Software Engineering 2: Object Oriented Software Engineering

Programming test – Autumn 2015

Max Cattafi (m.cattafi@imperial.ac.uk)

It's in your interest to read the whole document before writing any code.

Consider an application dealing with geometric entities and shapes on the cartesian plane. We will deal in particular with points, triangles and circles.

Points are identified by their coordinates. It should only be possible to instantiate them by passing actual coordinates (no default constructor should be available for points), they should have a member function returning the distance from another object point passed as argument (`sqrt` might require inclusion of `<cmath>`, if you have problems with the linker you can just return the square of the distance) and getters for their coordinates. No other member functions can be defined for points in this application.

In our application all shapes have a perimeter (indeed for circles it is usually known as circumference but it doesn't matter for us, it's still a perimeter). Triangles are identified by three points and it should only be possible to instantiate triangles by passing to their constructor objects of type point. Circles are identified by a point (representing the centre) and a radius, and it should only be possible to instantiate them by passing to the constructor an object of type point and a number for the radius. You can truncate π to 3.14.

Write a program which reads from two text files information about triangles and circles, stores this information in a suitable way and prints the perimeter of all the shapes.

For instance the file for triangles might look like this:

```
1.5 2.5 3 3 2 2
-0.5 1.2 1 1 -1 1
```

Each line represents one triangle by means of three pairs of coordinates, each pair of coordinates represents one point (a vertex).

While the file for circles might look like this:

```
-2 1.5 10
2 -1 5.3
```

Each line represents one circle by means of a pair of coordinates, representing one point (the centre), and a number for the radius.

Do not manually allocate dynamic memory with `new`. Store the triangles and the circles in two different lists (in the sense of `std::list`). After the input phase, create one (only one) suitable vector of pointers pointing to elements in both lists and use only this vector for the perimeter printing phase.

Notice that the files might contain inconsistent information, for instance representing a triangle with

```
0 0 0 0 0 0
```

which is clearly not a valid triangle.

In general in any triangle the length of each side must be less than the sum of the lengths of the other two sides. In the case of circles, the radius has to be greater than 0.

When an attempt is made to construct a triangle or a circle with inconsistent information, a `std::logic_error` exception (requires header `<stdexcept>`) should be thrown by the constructor, containing a message which identifies what kind of shape is throwing the exception and any other useful information.

In the main, during the input phase, the exception should be handled by interrupting the input, closing any file streams, printing an error message including the message contained in the exception. The execution should then go on printing the perimeter for all the shapes correctly constructed before the exception was thrown.

Some requirements to respect for the program design and architecture:

- Suitable relationships of inheritance and composition, encapsulation (appropriate visibility for class fields and avoiding getters and setters unless mentioned in the requirements), use of polymorphism.
- Use of initialization lists in constructors.
- Const-correctness and appropriate use of const references.

- Separation of source and header files.
- None of the functions (including member functions), except for the `main`, should contain user I/O (`cin` or `cout`) or file I/O in their implementation: I/O should only be in the `main`.
- Appropriate memory management.
- Use of iterators whenever possible and appropriate.