Advanced Programming in C++

Exercise – Geometric objects

Define classes to represent geometric objects. The following classes shall be defined:

Point
: Stores two coordinates, x and y, which shall be directly accessible (public members).

`Line`
: Stores the end points of a line as two coordinate pairs. The four coordinates shall be directly accessible (public). Line shall have a public member function returning the length of the line.

`Circle`
: Stores the radius of a circle (non-public). The radius, area, and perimeter are to be returned by member functions.

`Rectangle`
: Stores the two side lengths of the rectangle (non-public). The side lengths, area, and perimeter are to be returned by member functions.

`Triangle`
: Stores the three side lengths of the triangle (non-public). The side lengths, area, and perimeter are to be returned by member functions.

`Circular_Cylinder`
: Stores the circle radius and the height of the circular cylinder (non-public). The radius, cylinder height, area, and volume are to be returned by member functions.

`Rectangular_Prism`
: Stores the two side lengths of the rectangle and the height of the prism (non-public). The side lengths, height, area, and volume are to be returned by member functions.

`Triangular_Prism`
: Stores the three side lengths of the triangle and the height of the prism (non-public). The side lengths, height, area, and volume are to be returned by member functions.

Point an Line shall not be possible to derive from (to subclass).

Circular cylinder, rectangular prism and triangular prism are kind of three-dimensional equivalents to circle, rectangle and triangle, respectively. Let inheritance reflect this.

Circle, rectangle och triangle are two-dimensional (plane) objects, while circular cylinder, rectangular prism and triangular prism are three-dimensional objects (solids). The plane objects have the *area* and *perimeter* (profile). The solid objects have the *volume*, but also area and perimeter, where perimeter means the perimeter of the corresponding plane object. Define an interface (abstract class) for plane objects and an interface for solids, and let also these be base classes to the plane and solid objects, respectively (leads to multiple inheritance).

Each class shall have a default constructor, which, e.g., gives a Point the coordinates (0.0, 0.0), a Circle the radius l.0, etc.

There shall also a constructor for each type, to initialization an objects with appropriate initial values.

Operations for changing the state of the objects are not necessary to implement.

Every class shall implement an interface class Serializeable, having a virtual member function str(), which is supposed to return a string representation of the object in question in a serialized form. For a Point object, e.g., with identity number 9 and the coordinates (2, 7) the string "Point@9[x=2,y=7]" shall be generated and returned. Define Serializeable on a separate file Serializeable.h.

Every class shall implement an interface Cloneable, having a virtual member function clone(), which is supposed to create a dynamically allocated copy of the object in question and return a pointer to the new object. Define Cloneable on a separate file Cloneable.h.

Alternatively, let, for example, Serializeable implement the NVI pattern, with a public non-virtual member function str(), and a private virtual member function to_str().

Write a program which creates different geometric objects dynamically, and store the pointers to the objects in a vector. The program shall step through the vector, and for each object print which type of object it is and its characteristics (length, area, volume, etc.) and also its serialized form.

Before the program is terminated, the memory for the dynamic objects in the vector shall be released. Use a lambda expression or a function object and some suitable standard algorithm to do this.

These classes will not be especially interesting in practice and there are some inconsistencies, e.g that Point and Line stores coordinates, while the other do not. But several C++11 features are possible to use.

The main program can partly look as follows:

```
int main()
{
  vector<Geometric_Object*> objects;
  // Fill objects with different kind of geometric objects
  …
  // Traverse objects and print information for each object
  for (…)
  {
    // Copy (clone) the current object and if cloneable do the following
    cout << endl << "Type......: " << typeid(…).name() << endl;
    …
    if (line) {
      cout << "Length....: " << …->get_length() << endl;
    }
    …
    if (plane) {
      cout << "Area......: " << …->get_area() << endl;
      cout << "Profile...: " << …->get_profile() << endl;
    }
    …
    if (solid) {
      cout << "Volyme....: " << …->get_volume() << endl;
    }
    …
    if (serializeable) {
     cout << "Serialized: " << …->str() << endl;
    }
    // Destroy the copy
  }
  // Clear the vector
}
```

Possible extension: Let the program save the serialized descriptions for the objects on a text file and write a program that reads the text file an recreates the objects.