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 the line as two coordinate pairs. The four coordinates shall be directly accessible (public members). Line shall have a public member function for getting the length of the line.

`Circle`
: Stores the radius of the circle. The radius, area, and perimeter are only to be accessed through member functions.

`Rectangle`
: Stores the two side lengths of the rectangle. The side lengths, area, and perimeter are only to be accessed through member functions.

`Triangle`
: Stores the three side lengths of the triangle. The side lengths, area, and perimeter are only to be accessed through member functions.

`Circular_Cylinder`
: Stores the circle radius and the height of the circular cylinder. The radius, cylinder height, height, area, and volume are only to be accessed through member functions.

`Rectangular_Prism`
: Stores the two side lengths of the rectangle and the height of the prism. The side lengths, height, area, and volume are only to be accessed through member functions.

`Triangular_Prism`
: Stores the three side lengths of the triangle and the height of the prism. The side lengths, height, area, and volume are only to be accessed through member functions.

Each object shall have an identity number. The first object created shall have number 1, the second number 2, and so on.

Every class shall have a member function str() which returns a string representation of the object in a serialized form. For, e.g., a `Point` object with identity number 9 and the coordinates (2, 7) the string "`Point@9[x=2,y=7]`" shall be generated and returned.

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 characteristics *area* and *perimeter* (profile). The solid objects have the specific characteristic *volume*, but also area and perimeter, where we let perimeter mean the perimeter of the corresponding plane objektet. 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).

All objects shall be initialized by constructors. There shall be a default constructor for each object, which, e.g., gives a Point the coordinates (0, 0), a Circle the radius l, etc. There shall also be constructors for explicit intialization of objects.

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

Write a program which creates different geometric objects dynamically, and store the pointers to the objects in a `vector`. The program shall then step through the `vector` and for each object output 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. Create a function object `delete_object`, which can be used together with the algorithm `transform` to do this.

Note: These classes will not be especially interresting in practice and there are some inconsistencies, e.g that Point and Line stores coordinates, while the other do not.

The main program can partly look as follows:

```
int main()
{
  vector<Geometric_Object*>  objects;
  // Fill the vector with different kind of geometric objects
  …
  // Traverse the vector and output informatyion for each object
  for (…)
  {
    cout << endl << "Type......: " << typeid(…).name() << endl;
    …
    if (line)
    {
      cout << "Length....: " << line->get_length() << endl;
    }
    …
    if (plane)
    {
      cout << "Area......: " << plane->get_area() << endl;
      cout << "Profile...: " << plane->get_profile() << endl;
    }
    …
    if (solid)
    {
      cout << "Volyme....: " << solid->get_volume() << endl;
    }

    cout << "Serialized: " << (*iter)->to_string() << endl;
  }
  …
}
```

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.