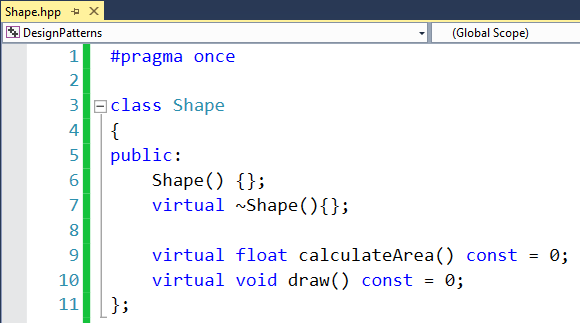
Creating a Shape Hierarchy

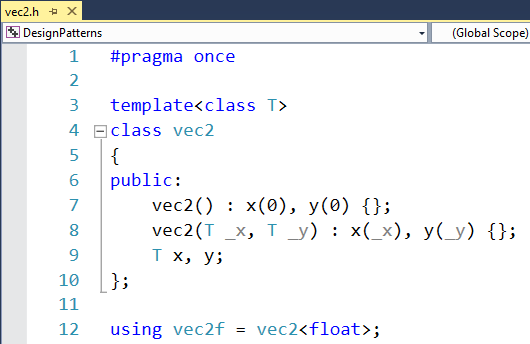
Our Application will have a simple Scene graph consisting of Rectangles and Circles. These two classes have common functionality. Namely they can be rendered (using OpenGL via cinder) and we can calculate the surface area.

# Create the Shape class

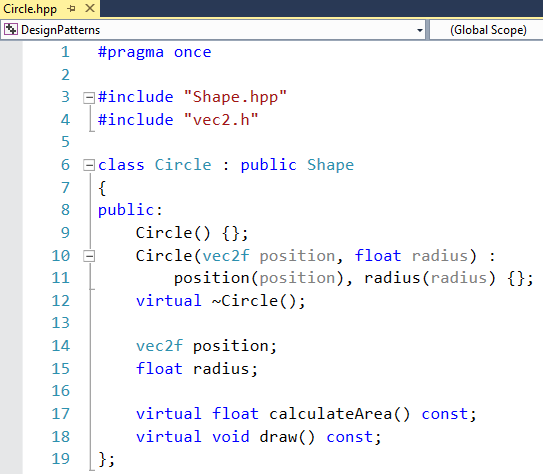
Add a new header file to the project named ‘Shape.hpp’. As Shape is an abstract concept, it will only hold pure virtual function calls as well as a virtual destructor. The virtual destructor is a best practice for base classes to ensure that a derived destructor is called if the object is deleted with a pointer of a base-class type. 

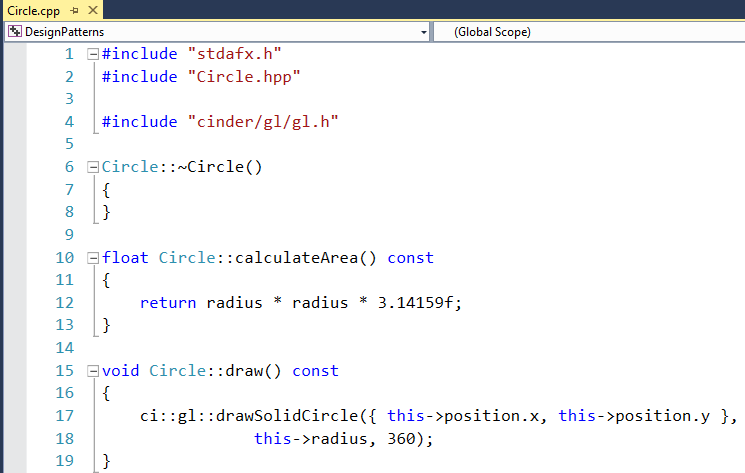
# Create the Circle class

The class circle derives from Shape and adds two properties, position and radius. The vec2f class is a template class specialized for float, as shown below:



Circle also implements the calculateArea and draw methods, which are implemented in the cpp.

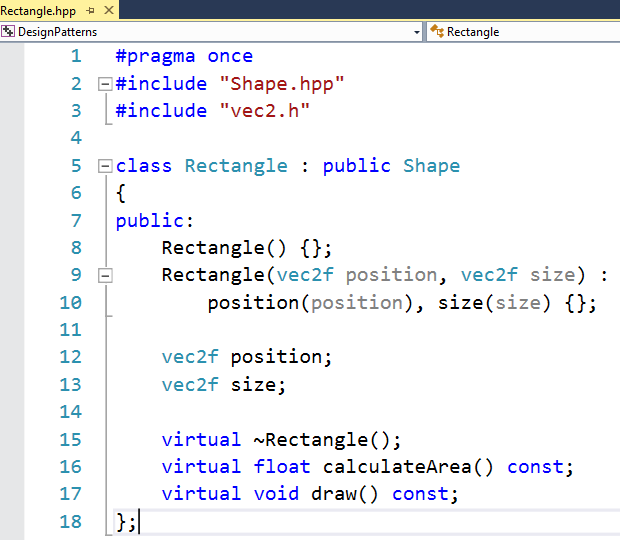


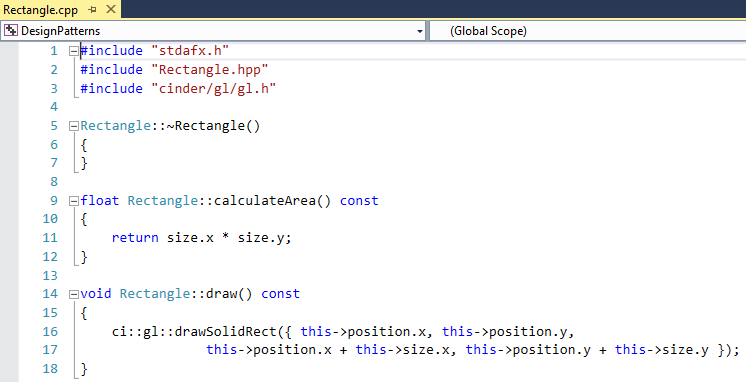


The drawSolidCircle method draws a circle at the given position and radius. The third argument defines the number of segments. The higher this number, the finer grained the circle is rendered.

# Create the Rectangle class

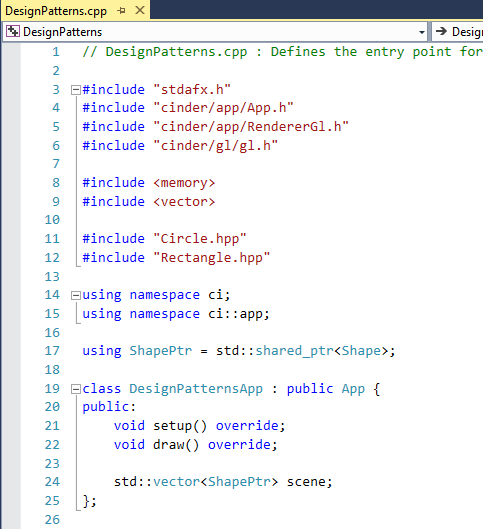
Very similar to the circle class, Rectangle extends the Shape class and adds two properties, the position as well as the size. It also implements the draw and calculateArea methods accordingly.



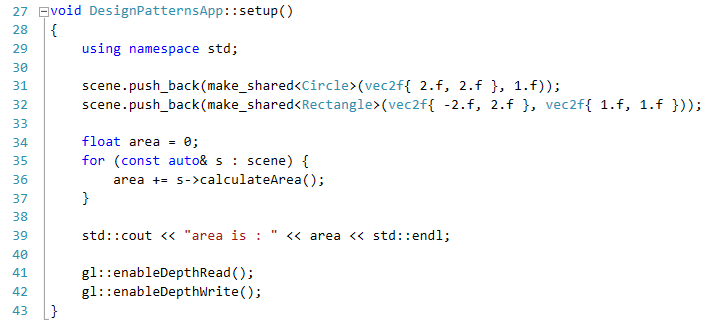


# Use the Shapes for constructing a trivial Scene

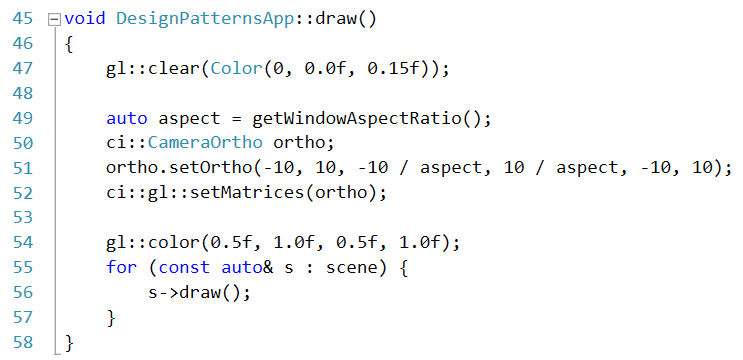
The scene will be an essential part of our application. It will hold all shapes in the scene. Since we want to use smart pointers for life-time management, we define an alias ‘ShapePtr’ that holds references to Shapes. We also add a vector of such pointers as a member Variable of our DesignPatternsApp class:



We also extend the setup method to construct a sample scene by adding a circle and a rectangle to the scene member variable. It also calculates the total surface area of the scene by aggregating the area of all shapes in the scene.



Finally we extend the draw method do setup an orthogonal camera (flat, 2D) that that sets the aspect ratio to reflect the window aspect ratio and a loop that iterates all shapes to draw them:



# Compile & Run

After compiling and running the application, two windows a console window and the actual output window should appear:

