Object Oriented Programming (CS1143)

Week 10

Department of Computer Science

Capital University of Science and Technology (CUST)

Outline

- Abstract Base Classes and Pure Virtual Functions
- Virtual Inheritance. Diamond Problem and its solution

Concrete Class and Abstract Class

- The classes we have designed so far are called concrete classes.
- A concrete class can be instantiated and create objects of its type.
- When we create a set of classes, sometimes we find that there is a list of behaviors that are identical to all classes.
- For example, assume we define two classes named Rectangle and Square. Both of these classes have at least two common behaviors: getArea() and getPerimeter().
- How can we force the creator of these two classes to provide the definition of both member functions for each class?
- We know that the formulas to find the area and perimeter of these geometrical shapes differ, which means that each class must create its own version of getArea and getPerimeter.

Abstract Class

- The solution in object-oriented programming is to create an abstract class, which forces the designers of all derived classes to add these two definitions to their classes.
- An abstract class is a class with at least one pure virtual function.

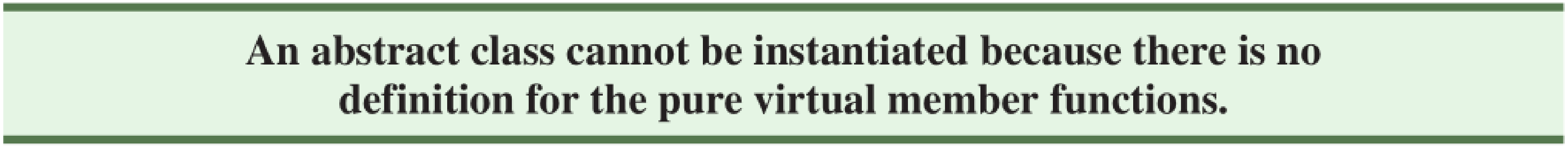
An *abstract class* is a class with at least one *pure virtual function*.

Pure Virtual Function

- A pure virtual function is a virtual function in which the declaration is set to zero and there is no definition in the abstract class.
    - **virtual double getArea() =0;**

No Instantiation of Objects for abstract class

- For an object to be instantiated from a class, the class must have the definitions of all member functions.
- We cannot instantiate an object from an abstract class because it does not have the definition of its pure virtual functions.



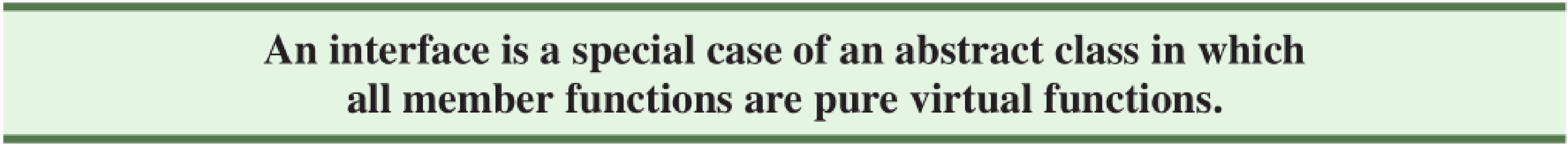
An abstract class cannot be instantiated because there is no definition for the pure virtual member functions.

Definition of Pure Virtual Functions

- The abstract class does not define its pure virtual function, but every class that inherits from the abstract class must provide the definition of each pure virtual function or declare it as a pure virtual to be defined in the next lower level of the hierarchy.
- **If a derived class does not provide the definition of pure virtual functions and also does not declare them as pure virtual, it also becomes abstract. It will not give any error but it means now you cannot create an object of this class also.**

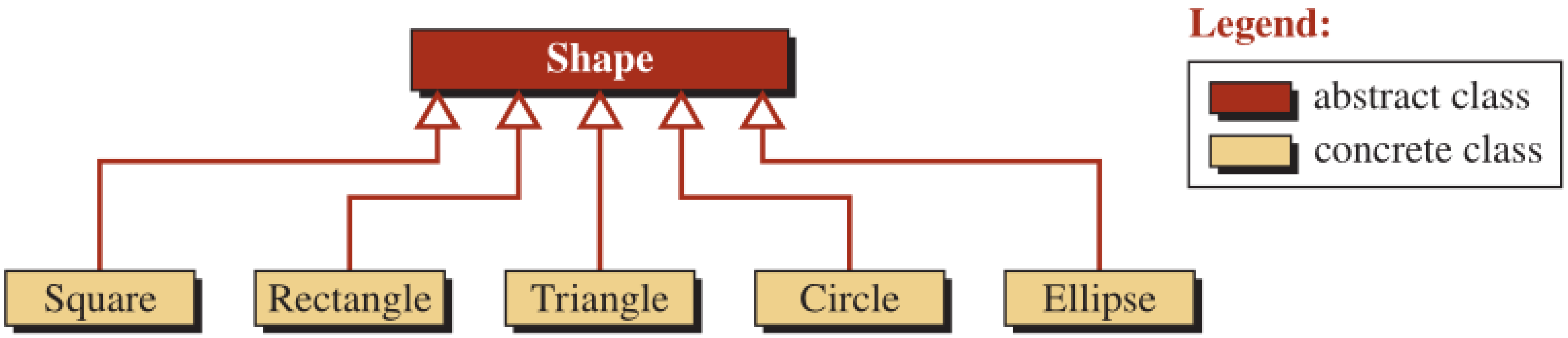
Interfaces

- An abstract class can have both virtual and pure virtual functions.
- In some cases, however, we may need to create a blueprint for inherited classes.
- We can define a class with all pure virtual functions.
- This class is sometimes referred to as an interface; we cannot create any implementation file from this class, only the interface file.



An interface is a special case of an abstract class in which all member functions are pure virtual functions.

Example: Shape Class



Shape

Square

Rectangle

Triangle

Circle

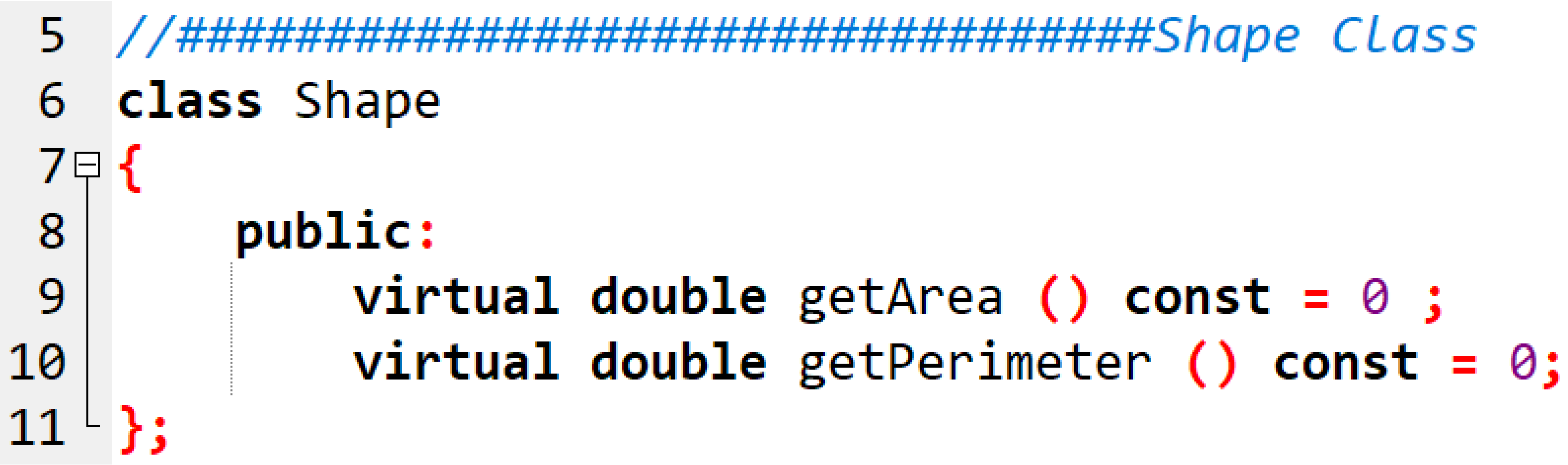
Ellipse

Legend:

🟥 abstract class

🟨 concrete class

Shape Class

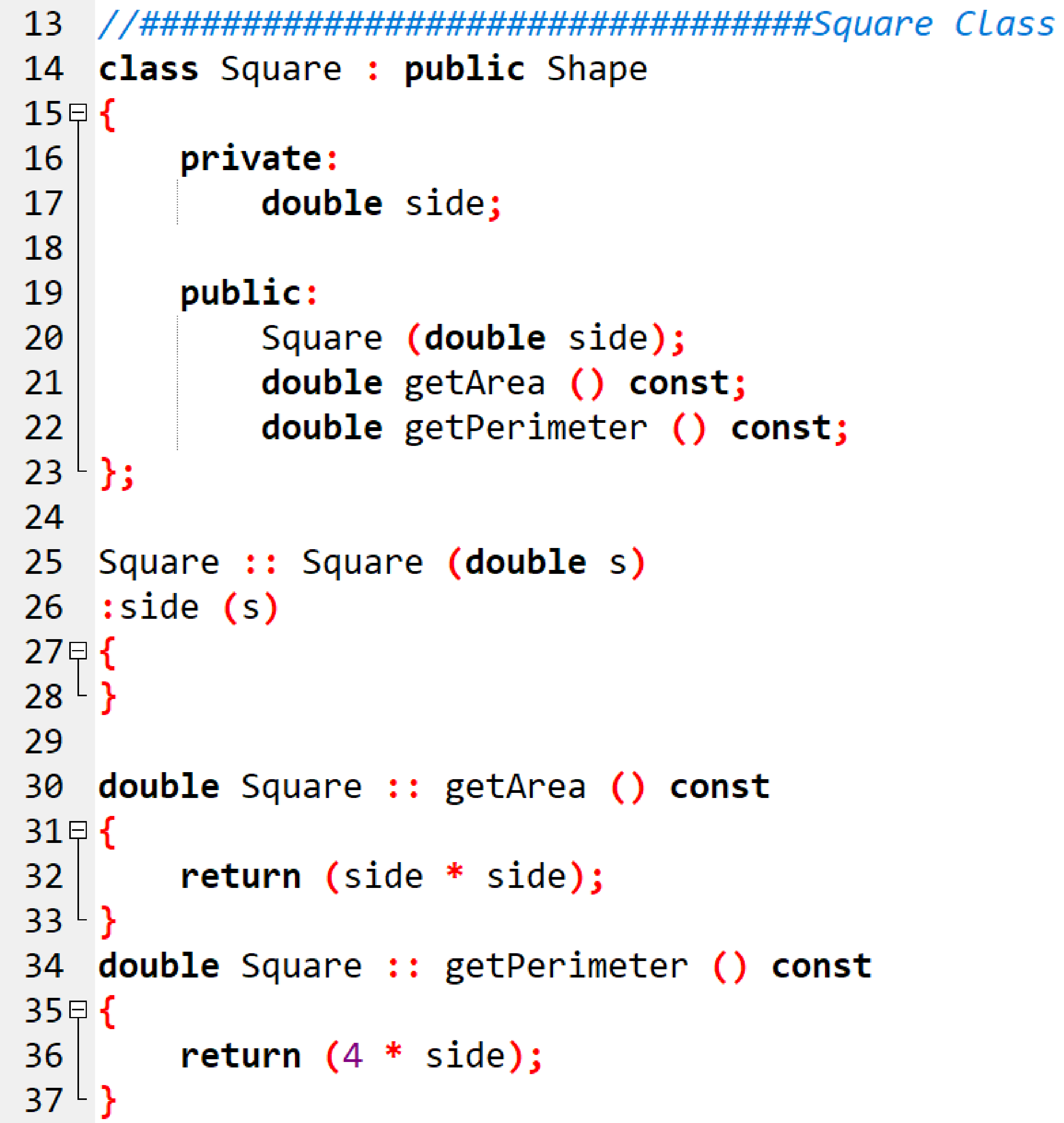


```
//#################################Shape Class
class Shape
{
    public:
        virtual double getArea () const = 0 ;
        virtual double getPerimeter () const = 0;
};
```

Description

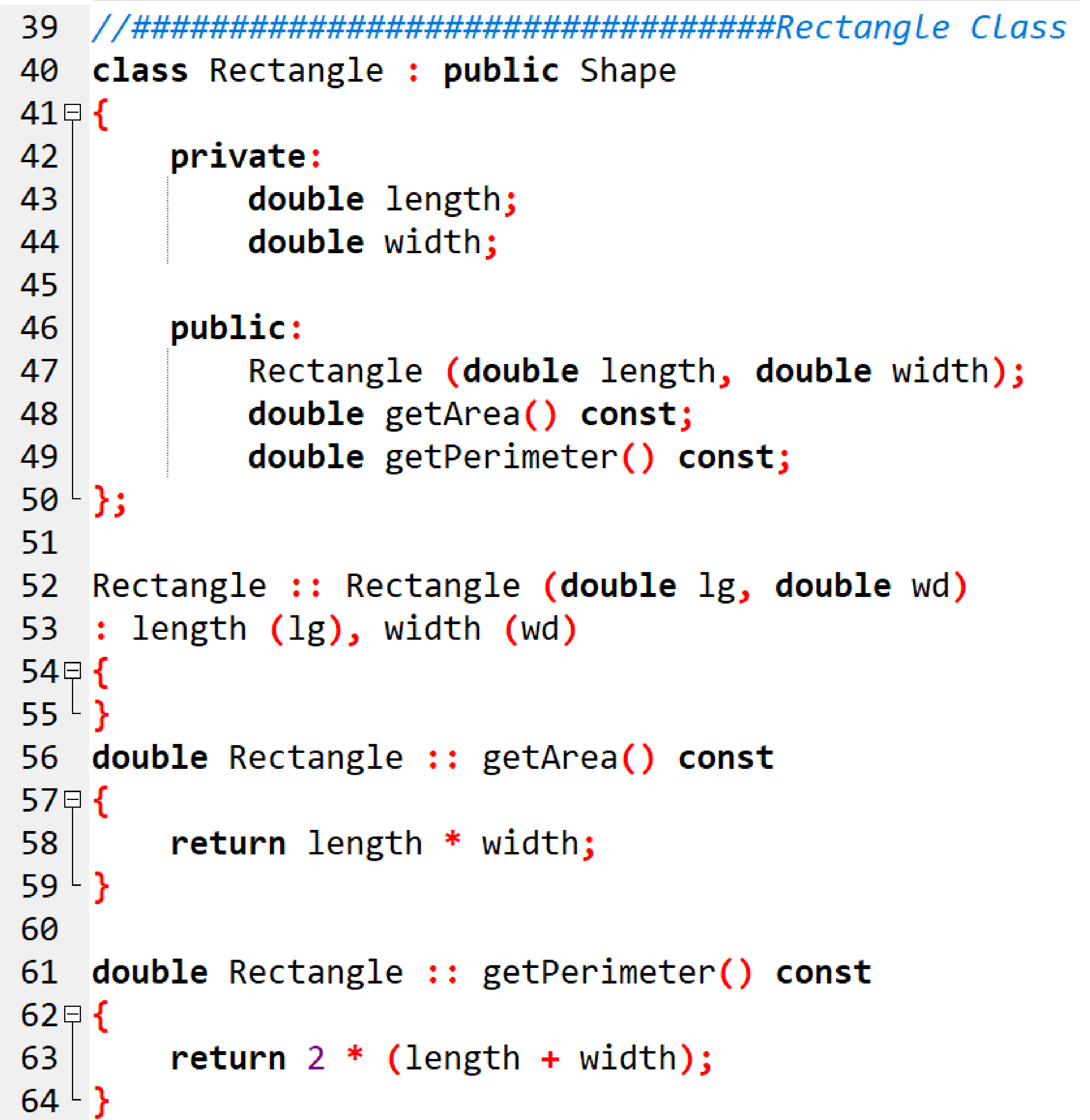
- Note that we have no data members and only pure virtual member functions in this program.
- The reason is that we do not want objects instantiated from this class; its purpose is only to force the derived classes to implement the pure virtual member functions.
- The two pure member functions force each derived class to have at least two member functions to calculate the area and perimeter of the corresponding shape.

Square



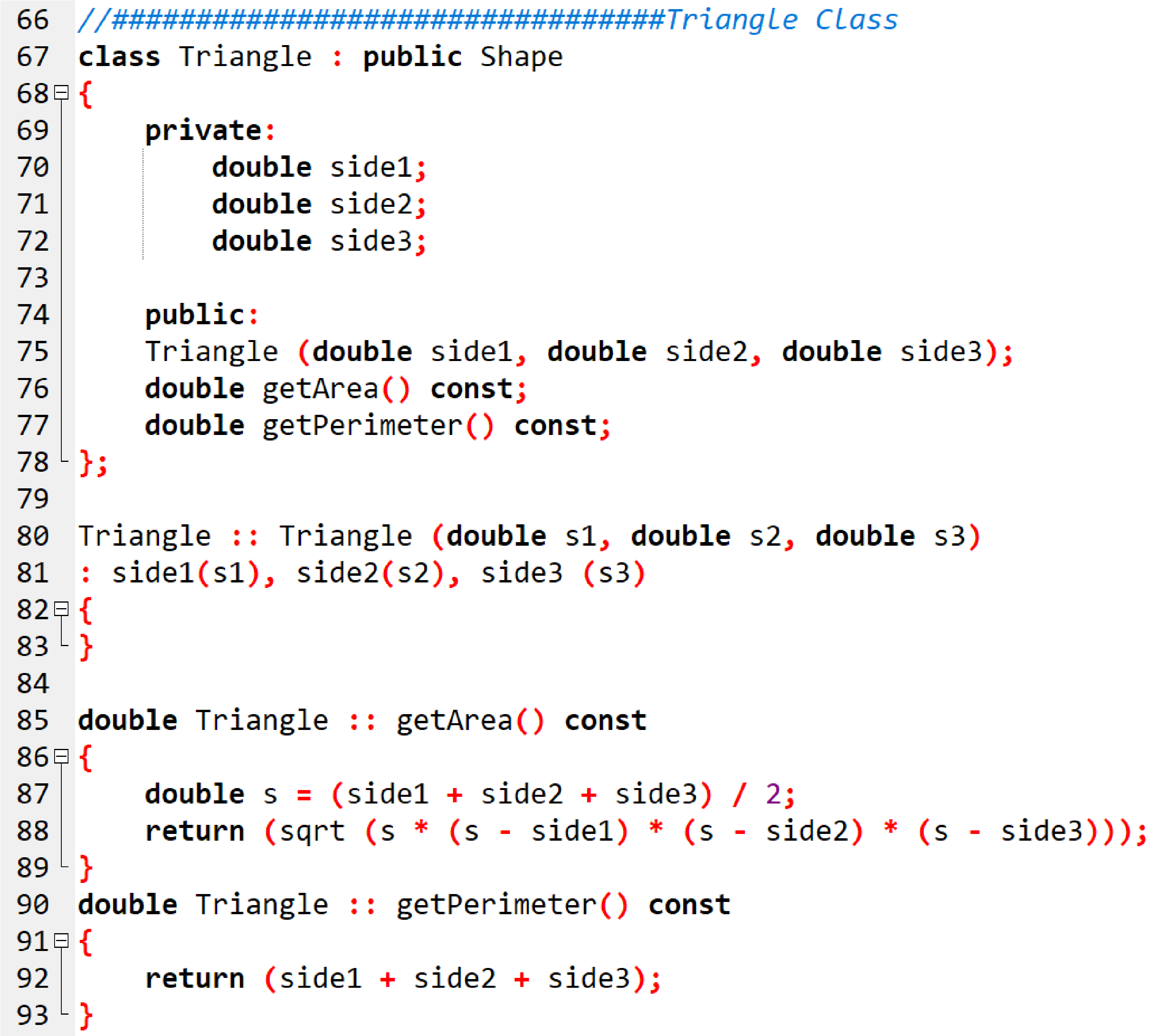
```
//#################################Square Class
class Square : public Shape
{
    private:
        double side;

    public:
        Square (double side);
        double getArea () const;
        double getPerimeter () const;
};

Square :: Square (double s)
:side (s)
{
}

double Square :: getArea () const
{
    return (side * side);
}
double Square :: getPerimeter () const
{
    return (4 * side);
}
```

Rectangle



```cpp
//#################################Rectangle Class
class Rectangle : public Shape
{
    private:
        double length;
        double width;

    public:
        Rectangle (double length, double width);
        double getArea() const;
        double getPerimeter() const;
};

Rectangle :: Rectangle (double lg, double wd)
: length (lg), width (wd)
{
}
double Rectangle :: getArea() const
{
    return length * width;
}

double Rectangle :: getPerimeter() const
{
    return 2 * (length + width);
}
```

Triangle



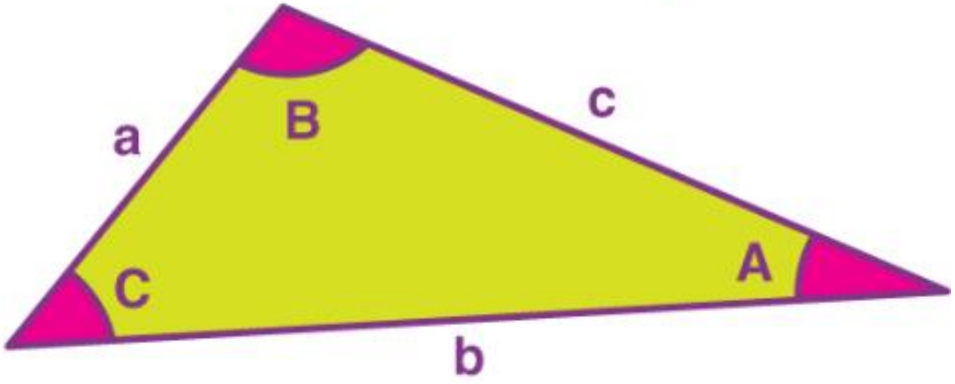
```cpp
//###################################Triangle Class
class Triangle : public Shape
{
    private:
        double side1;
        double side2;
        double side3;

    public:
    Triangle (double side1, double side2, double side3);
    double getArea() const;
    double getPerimeter() const;
};

Triangle :: Triangle (double s1, double s2, double s3)
: side1(s1), side2(s2), side3 (s3)
{
}

double Triangle :: getArea() const
{
    double s = (side1 + side2 + side3) / 2;
    return (sqrt (s * (s - side1) * (s - side2) * (s - side3)));
}
double Triangle :: getPerimeter() const
{
    return (side1 + side2 + side3);
}
```

Area of triangle with 3 sides

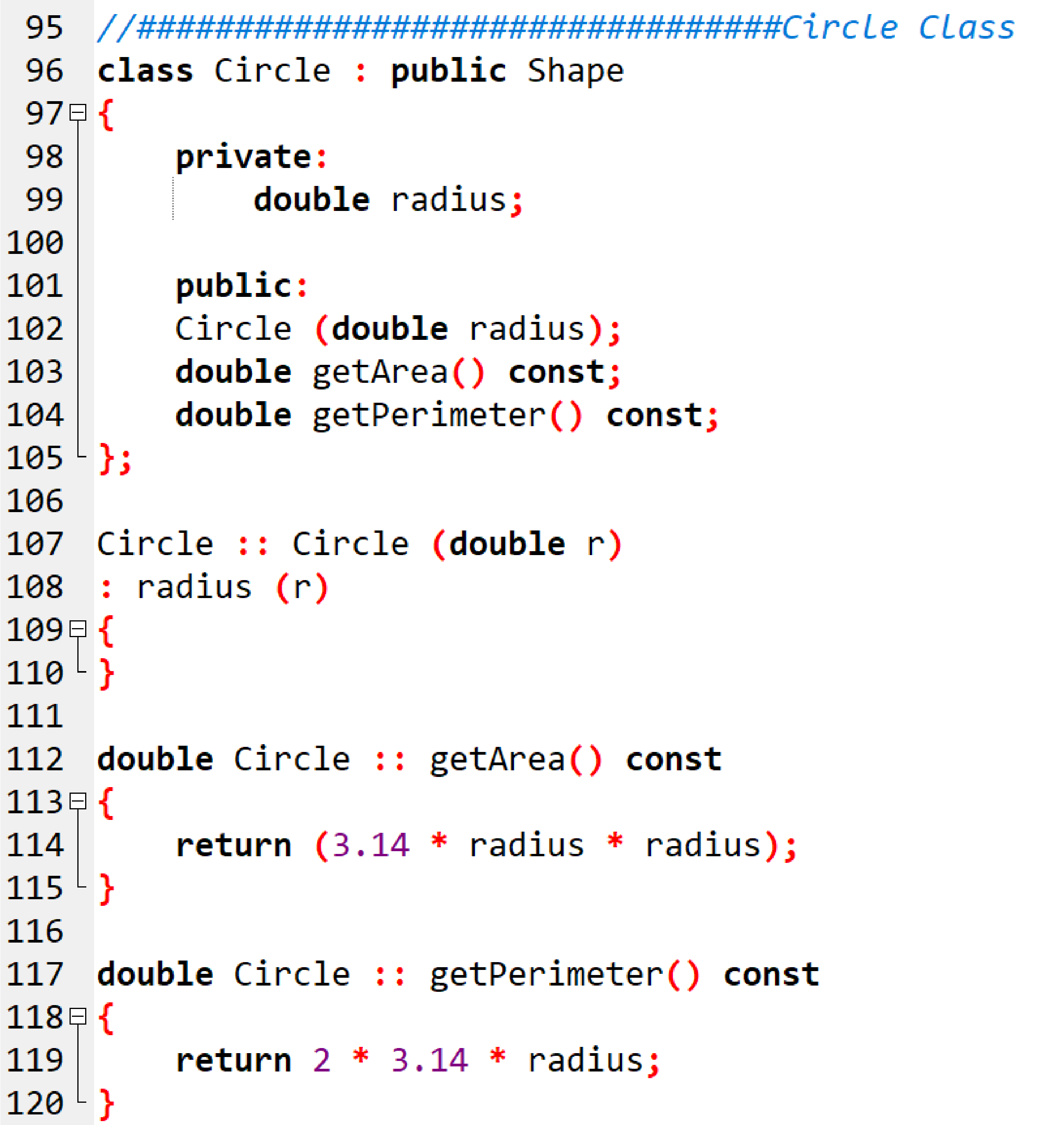
- https://byjus.com/maths/area-of-triangle-with-3-sides/

$$\text{Area, } A = \sqrt{s(s-a)(s-b)(s-c)}$$

Where,
$S = \text{Semi perimeter} = \dfrac{a+b+c}{2}$

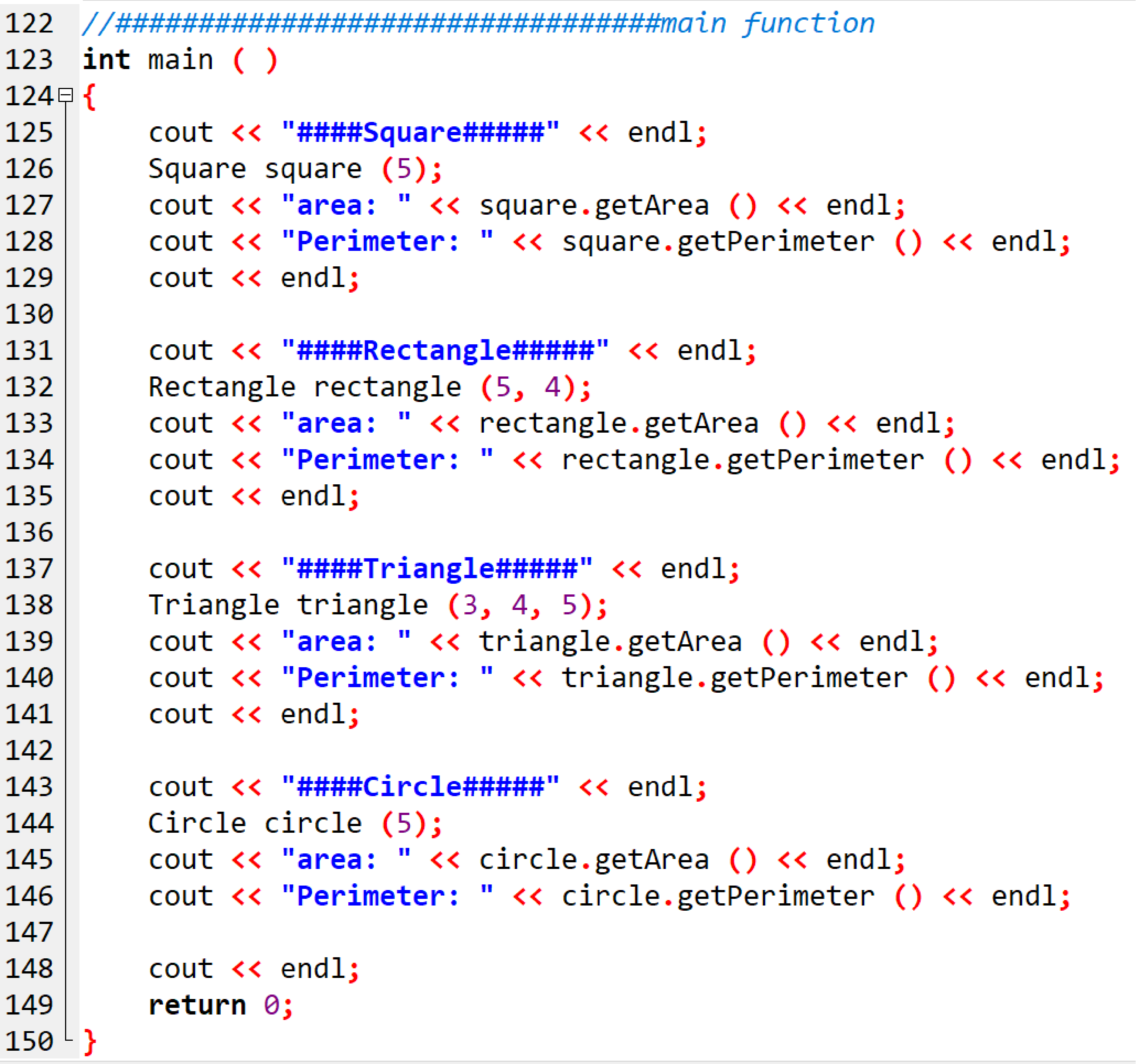


Circle



```
//##################################Circle Class
class Circle : public Shape
{
    private:
        double radius;

    public:
    Circle (double radius);
    double getArea() const;
    double getPerimeter() const;
};

Circle :: Circle (double r)
: radius (r)
{
}

double Circle :: getArea() const
{
    return (3.14 * radius * radius);
}

double Circle :: getPerimeter() const
{
    return 2 * 3.14 * radius;
}
```

Main



```
//####################################main function
int main ( )
{
    cout << "####Square#####" << endl;
    Square square (5);
    cout << "area: " << square.getArea () << endl;
    cout << "Perimeter: " << square.getPerimeter () << endl;
    cout << endl;

    cout << "####Rectangle#####" << endl;
    Rectangle rectangle (5, 4);
    cout << "area: " << rectangle.getArea () << endl;
    cout << "Perimeter: " << rectangle.getPerimeter () << endl;
    cout << endl;

    cout << "####Triangle#####" << endl;
    Triangle triangle (3, 4, 5);
    cout << "area: " << triangle.getArea () << endl;
    cout << "Perimeter: " << triangle.getPerimeter () << endl;
    cout << endl;

    cout << "####Circle#####" << endl;
    Circle circle (5);
    cout << "area: " << circle.getArea () << endl;
    cout << "Perimeter: " << circle.getPerimeter () << endl;

    cout << endl;
    return 0;
}
```