Dylan Brown

09/04/19

CS-250

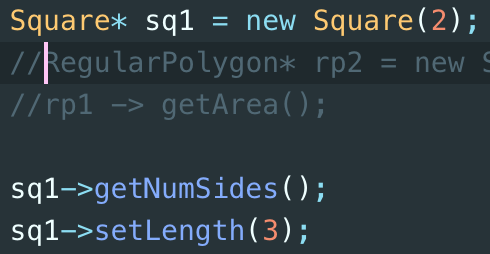
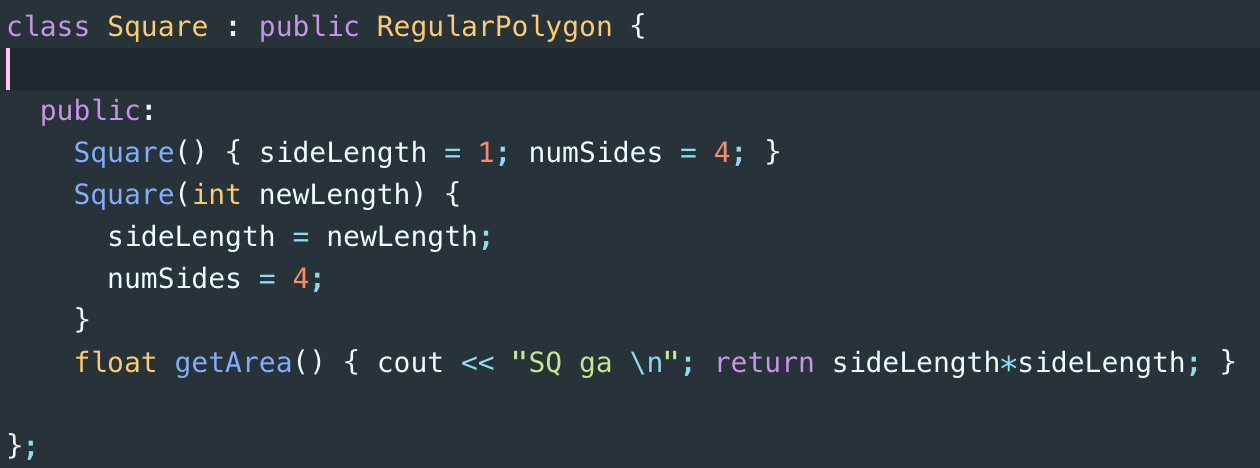
Rosasco

A Writing Assignment

Object-orientated programming is a programming model where the program is based around different objects that have data and functions. An object, or sometimes called a class, has member data known as attributes and functions known as methods. The most popular programming languages that tend to follow object-orientated programming are Java, C++, and Ruby. This writing assignment aims to analyze a program written in C++. This program was written in the early spring of 2019. The program explores **classes**, objects, subclasses and **inheritance**, and **overriding** methods. It was written in C++, it uses constructors to create new shapes, and uses a get area function to calculate the area.

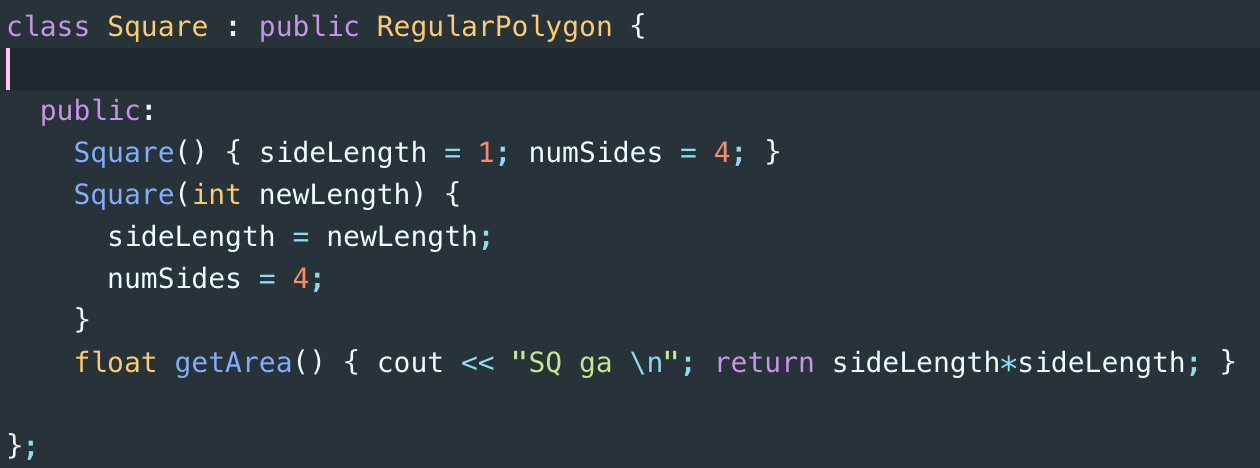
 First, let us explore classes in Polygon.cpp. Classes are blueprints for objects. Classes include trusted subroutines which track with what type of object it is. Moreover, it has members who keep track of data regarding the class. These members can be public, protected, or private. Polygon.cpp has 3 classes – Regular Polygon, Square, and Hexagon. Regular Polygon is listed below.

It describes a blueprint for a regular polygon shape. It has members that account for side length and the number of sides. It also has default setters and getters for its members, which is typical in an object-orientated model. It has a constructor that initializes these members and it has a blueprint for writing an area function. This function is abstract because it does not define how it should be used; that is for its children to decide. This code snippet represents a class because of these things. It has members that can save data regarding important information on the object. It has methods that can be called on objects to make them “do things”. Lastly, it is can be replicated and extended to different subclasses.

 Next, let us explore inheritance. Inheritance establishes a relationship between two or more classes. First, we have a “parent” or superclass. This is a normal class that defines members and methods. Then, we have a “child” or subclass. This class extends its parent class. In simpler terms, the child class receives members and methods from the parent class (e.g. parents passing down DNA to their child). These methods can be overridden and overloaded, but this is something that we will go into later. Inheritance describes an “is-a” relationship. Listed below is a class, Square, an instantiation of a Square object, and some methods.

We can classify a square through this concept of an “is-a” relationship. We say that Square is a Regular Polygon. C++ creates subclass by using the “:” operator. When we create a new square object, we have access to the methods that its parent created (as long as we didn’t override them). When we call getNumSides() and setLength(), the program returns the float and integer that was described from its parent class. This allows us to encapsulate data that is in parent classes and only allow child classes to see this data. Inheritance is essential to a well-written objected orientated program because it allows us to easily reuse code.

Lastly, let us examine overriding methods. Overriding takes place when a child class implements a method differently than it is stated in its parent. This is useful when we implement object-orientated programs in the real world. When classes are considered related to some degree, we can one a subclass of the other. Then, we just need to change the methods that are different. Overriding is also necessary when we use abstraction in our programs. Child members sometimes have to define how to use certain methods. This concept makes our program reproducible and reusable. Listed below is the square class from Polygon.cpp.



In the Square class, we have to override the getArea() function. This is because the parent class never defines it. This makes our Regular Polygon class a template for other classes. We can use it as a parent class and implement the getArea() function on all regular polygon shapes. Our code is then easier to read and more versatile.

Lastly, let us quickly touch on the difference between objects and classes. An object is an instance of a class. Classes are simply just the blueprint for an object. This is important because we can hand other programmers our classes and leave the implementation, and even subclasses, up to them. Object-orientated design allows us to do this, and this is why we learn and study it.