

Figure 1 - Horse derived class test

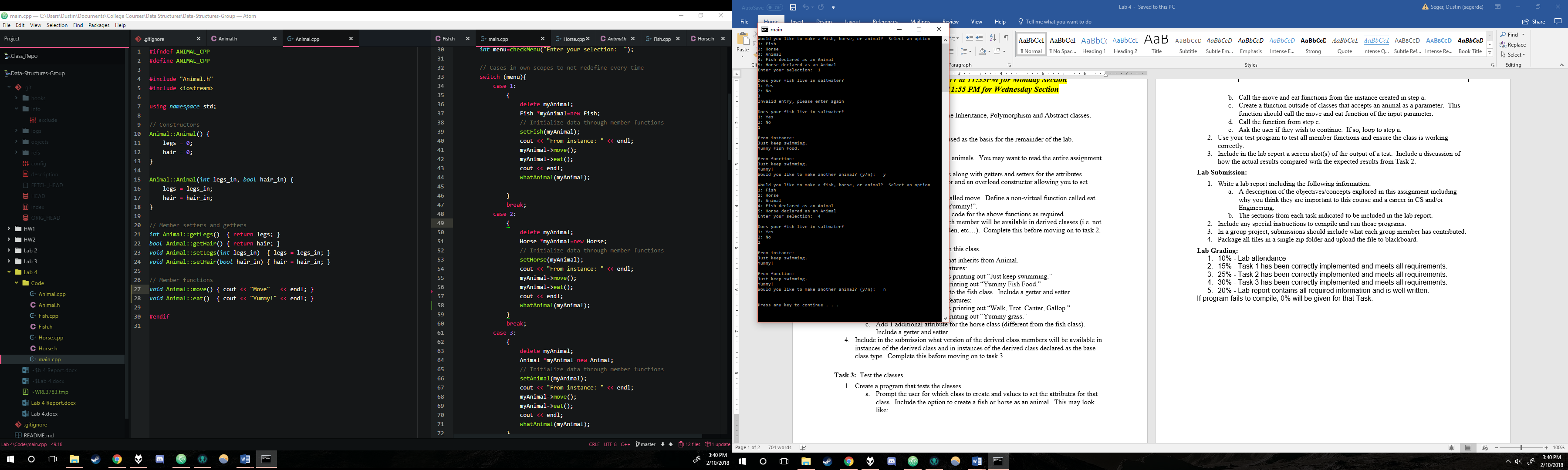


Figure 2 - Fish derived class test

**Inheritance Hypothesis vs. Results**

Based on our knowledge of class inheritance, a hypothesis can be made as to what version of the derived class members will be available in derived class instances versus instances declared as the base class type. All derived classes using the public access modifier should inherit all the members of the Animal class since they are all either protected or public. Dedicated derived class instances have the move function overridden through dynamic binding and the eat function redefined through static binding. Because of this, they will have access to those newly defined functions as opposed to the originals in the Animal class. They also should have access to the Animal class’ original members legs and hair, along with the getters and setters there, as well as the derived class’ own members, getters, and setters. A derived class defined as a base class type should function relatively similarly, only that the derived class’ members are not accessible directly. The eat function, although redefined, should call the original Animal class’ version since it is statically bound. However, the move function is dynamically bound, meaning that the new derived class version should be called despite it being defined as the base class.

After implementing and testing the classes, our original hypothesis can be confirmed. Much of the hypothesis was based on class notes and discussions on inheritance, and it seems to prove true in practice as well. By using our program to select the correct menu items to initialize different classes in different ways, it was clear to see that everything checked out. The biggest deciding factor was the statically bound eat and dynamically bound move functions. If a derived class is defined directly, it does indeed access its own functions. However, defining a derived class as a base Animal class proved that the dynamically overridden function was called, but the redefined eat function was instead called from the original class. After defining a derived class as an Animal base class type, it was also possible to initialize the new derived class values through a constructor, but there is no way to access them after the fact.

**Objectives / Concepts Explored**

The concepts explored in this lab were class inheritance and polymorphism. Inheritance was explored through the use of derived classes, while class polymorphism was explored through using static and dynamic binding within the base and derived classes. Both of these concepts are important as a unit because they are so crucial in object-oriented programming, which is a baseline in almost all C++ careers and courses. Inheritance is useful for the sake of efficiency and code cleanliness when dealing with multiple similar but unique classes. Without inheritance, every class would need to be defined separately, regardless of how alike it is to another class. This would inevitably result in repetitive and lengthy code, especially in large programs with a multitude of classes. Polymorphism within classes is important when dealing with inheritance in the case that the derived class does something differently. In this way, you can call the same function from two different classes of similar derivation, and they will do two different things if the programmer wishes. This cleans up code in the main and allows for a more versatile program in any C++ course or career.

**Compile Instructions**

Compile using G++ for Windows with default settings. All files should be in the same directory.

**Group Member Contributions**

Zach created the derived Fish and Horse classes, Dustin created the base Animal class. The work on main was divided evenly between us through high collaboration.