Kyle O’Connor

Smit Patel

Saylee Dharne

Evan Akers

Lab Report 4: Inheritance, Polymorphism and Abstract classes

This lab explores the concepts of Inheritance, Polymorphism and Abstract classes. These concepts are import to CS and Engineering when one makes programs larger than a single task. The idea of deriving classes is very convenient when trying to make or maintain larger projects. Making two similar classes with similar code is redundant, you end up having dozens of classes, you have many lines of similar code. Changing the code across classes is redundant and time consuming. Using base and derived classes and understanding the concepts associated with them, saves time and memory. Being able to make a base class that all derived classes can use means you only should write that code once. With that, understanding how member functions that are virtual and those that are not virtual is important to designing the base and derived classes.

Task 1

We predict that all of the attributes and functions declared as public in the base class would be available in any derived classes and to any other calls to those functions and attributes. We also predict that the data attributes declared as protected will be accessible to derived classes if they are not overwritten in the derived class. The protected members would be inaccessible to everything other than the base class itself and any derived classes. The move function specifically will be available to the derived class if it is not overwritten, because it is a virtual function, so the override preference would be given to the derived class.

*Table 1: Table of Access to Derived Classes*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Member functions and variables | Age | getAge() | setAge() | Name | getName() | setName() | Animal() | Animal(string name, float age) | eat() | move() |
| Access level | Protected  Available | Public  Available | Public  Available | Protected  Available | Public  Available | Public  Available | Public  Available | Public  Available | Public  Available | Public  Available if not overwritten |

Task 2

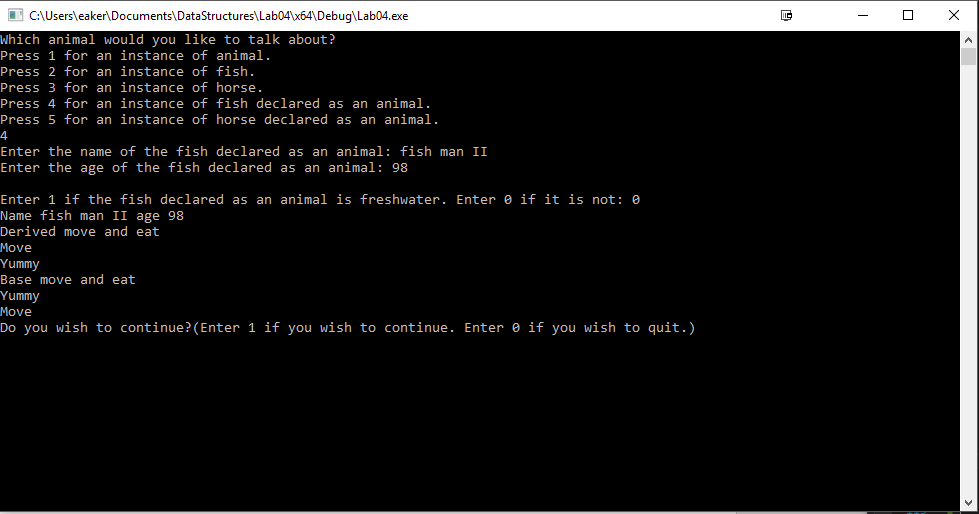
For the member variables and functions associated with age and name, we predicted that they would reference the base class because they are not overridden in the derived classes. Derived specific attributes like isMini or freshwater would be referenced to the derived class because they are not available to the base class. We also predicted that derived classes declared as animals would call base function eat, but not move because it is virtual.

*Table 2: Tables of Classes Called*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Member functions and variables | Age | Getters and setters for age | name | Getters and setters for name | eat() | move() | isMini | Getters and setters for isMini | freshWater | Getters and setters for freshWater |
| Derived Class- Horse | Base | Base | Base | Base | Derived | Derived | Derived | Derived | N/A | N/A |
| Derived Class- Fish | Base | Base | Base | Base | Derived | Derived | N/A | N/A | Derived | Derived |
| Base Class- Horse | Base | Base | Base | Base | Base | Derived | Derived | Derived | N/A | N/A |
| Base Class- Fish | Base | Base | Base | Base | Base | Derived | N/A | N/A | Derived | Derived |

Task 3

*Figure 1: Task 3 Output*



Our predictions for Task 1 were correct as the derived classes could access base class attributes and functions. The predicted functions and members from the base class were available to the derived class, although they could be overwritten by the derived class.

Our predictions on the most part for Task 2 were correct. Our previous logic seems to have been applied correctly to this scenario. One thing we did not predict was that the move function that was used when a fish was declared as an animal. We thought that because it was a virtual function that the derived class would be called, but the base class move function was called instead. This is because if a fish or horse is declared as an animal it does not have access to the derived move function so it calls the base class move. We also expected the derived member variables and functions to be available to derived classes declared as animals but that did not compile because the base class can’t reference the derived members. All other predictions met our expectations

Compilation Instructions

This has been tested by creating a new project within Visual Studios with the following options:

Win32 Console Application

Create directory for solution OFF

Empty project ON

Precompiled header OFF

SDL OFF

Then:

Add the following files to the projext:

1. TestProgram.cpp
2. Horse.cpp to the project
3. Horse.h to the project
4. Fish.cpp to the project
5. Fish.h to the project

Build and run.

Contribution of Team Members

Amongst the individual functions,

Kyle O’Connor did task 1

Saylee Dharne did task 2 and edited the lab report

Evan Akers did part of task 2, fixing the derived declarations as animal, and the lab report

Smit Patel did task 3