Kristian Snyder

Data Structures CS2021 Section 001

Prof. Charles Zimmer, Prof. Karthik Vadambacheri

Lab 04

This assignment investigated inheritance and how some uses of it, combined with polymorphism, can result in unexpected program execution. Specifically, the differences in what function would be executed depending on whether the instance is derived and the function is declared virtual were explored. Many of the ways in which the C++ compiler and runtime decide which function to call depending on the occasion are not at all obvious and can result in difficult-to-find bugs in many programs. By catching these quickly and avoiding confusion, it shows a deeper understanding of the C++ language and implementation and is invaluable in a professional position when building scalable, maintainable software. Small bugs like this are commonly extremely difficult to catch, as the program will continue execution without complaint and the difference in result from what would be expected could take a long time to find.

While the decisions that the C++ compiler and runtime make when evaluating polymorphic behavior and inheritance are deterministic and completely logical, they can behave in unpredictable ways to someone with little experience and so should be studied carefully to prevent future misuse. They are some of the most powerful tools of C++ and other object-oriented programming languages and enable enormous capability, but need to be treated with care when designing classes to prevent unexpected behavior.

**Tasks 1 and 2:**

*Hypotheses* of availability of attributes to derived classes, version of attributes available in derived classes

**For Animal:**

The attributes will only be available to derived classes through their getters and setters. The actual variables will be *unavailable* to all derived classes.

move() and eat() are available to instances of a derived class unless overridden in all cases, as they are public. *(Note: as explained in the analysis of task 3, how these functions are available to derived instances differs due to move() declared as virtual.)*

**For Fish and Horse (referred to as derived):**

The attributes in Animal are available only through Animal’s getters and setters, which are passed down to the derived instances.

move() is implemented by Fish and Horse, so calls to move() through an instance will pass to the derived implementation. (Just keep swimming. or Walk, Trot, Canter, Gallop. will be printed.) Calling move() as a derived class with an Animal pointer will result in the base class’s implementation being called. *(This is wrong, with a correction and an explanation why below.)*

eat() is overridden by the derived classes, so calls to eat() will print Yummy Fish Food. or Yummy grass. instead of Yummy! as defined in Animal. For derived instances with Animal pointers, calls to eat() will print out Yummy! instead.

**When running main() (task 3):**

In task 2, I hypothesized that calling move() with a Fish or Horse acting as an Animal would result in calling the base’s class’s function. Instead, the virtual function move() *always* called the derived version of the function instead of that defined in Animal. This was due to the function being virtual: at runtime all instances of the class have the closest version of the function stored in a lookup table for the program to use when it is virtual. In this case, any derived instances that override an inherited function will have the derived implementation stored, not the base implementation. This is called *late* or *dynamic* *binding.*

The function call of eat() proceeded as expected; each call by an instance of any class called the Animal implementation, printing Yummy! This is due to *static binding* of eat(), as it is not a virtual function and so does not use dynamic binding. Therefore, at compile time, the call to eat() will always be to the implementation of Animal; the parameter passed to the function callMoveEat(Animal \*animal) is assumed to be the base class at compile time since the compiler cannot know what is passed to the function points to an Animal or a derived class instance..