CS 162, Fall 2016: Group 15 Activity, Polymorphism

Group Members who Contributed:
Peter Moldenhauer
Jessica Wade
Joel Yoder
Clifford Cho
Mike Mann
Michael Kozlowski

Group member who did not contribute: Philip Sigilito

1. What is the slicing problem?

The slicing problem occurs when an object of a derived class is copied to an object of the base class using static binding. This means that the compiler will only use information available at compile time, which will not include any information specific to a derived class. Therefore, when an object of a parent class is assigned the value(s) of a child class, the child class inherits the properties of the parent class, but the parent class cannot directly inherit the properties of the child class. The base class copy will not contain any of the additional variables that are defined in the derived class. So this data that was previously in the derived class has been "sliced off" and cannot be accessed.

For example, `class Employee` creates an object `emp` and `class Hourly::Employee` creates an object `hrlyEmp` and if `emp = hrlyEmp;` any of the properties that make `hrlyEmp` a special kind of `Employee` (an `Hourly` one) would not be passed on to `emp.` They would be sliced off, leaving `emp` with just the properties of `Employee.`

2. Why is it (the slicing problem) a problem?

Essentially, without knowledge of the slicing problem, the derived classes can lose their specialization. Any unique members of the derived class can get lost, and therefore the derived class objects would lose their uniqueness.

The slicing problem can result in a programmer losing information if an object of a derived class is assigned to an object of the base class that has already been instantiated. If objects are assigned to each other in this manner, someone may think that the copies are successful but they actually are not. Only PART of the copy is successful. Only the parts of the derived class that are inherited from the base class are copied over. The additional parts of the derived class are not inherited from the base class, and therefore are not copied.

Slicing information without knowing it will also lead to errors and can be a pain to find and correct. This can lead to memory leaks and corrupt memory. For example, if the derived class creates a dynamic array, but then is sliced into it's base class, it will be unable to access the destructor in the derived class to deallocate the memory. This will lead to memory leaks, which can possibly lead to memory corruption. Also, this would cause issues if any derived class function was supposed to be called or the program needed to access the derived class's member variables that were lost.

3. How does polymorphism solve the slicing problem?

Without polymorphism, any objects that are derived from the base class will be "locked" into what is defined in a base class object. When static binding is performed at compile-time, the compiler mandates use of only the base class functions. Polymorphism utilizes dynamic binding, which is done at run-time, so any dynamically allocated pointers can access the appropriate functions whether it be the base class functions or the overridden derived class functions. This means that the derived class object's unique member variables and functions remain accessible.

It is also important to note that in order for the compiler to read the polymorphic behavior, it requires the program to use a pointer or reference to access the derived object. For example, if a base class A object was created, it can only contain member variables defined in class A. However, if it is a pointer to class A, then the pointer can point to any object derived from class A. So, if class B was derived from class A, even if the pointer is a pointer to class A, class B's unique functions will still be accessible through the pointer.

In addition to the slicing problem, polymorphism also solves the problem of defaulting to the base class version of a member function when a derived object with a more specialized version of the same function is created. This is done through declaring functions with the keyword "virtual." Specifically, whenever you make a function "virtual" in a class, this indicates that you are using polymorphism. When virtual functions are used, C++ will look in each of the specific derived classes to use the appropriate function instead of simply using the function in the base class. This way, additional information will not be ignored because C++ will know which specific function to use and not simply just use the more general base class version.

In short, the way that polymorphism solves problems such as the slicing problem and the function override problem is through the use of virtual functions and dynamic memory allocation of derived class objects, which can be accessed by pointers to the base class.