

## CSE 332 Studio Session on C++ Inheritance Polymorphism

These studio exercises are intended to introduce different forms of inheritance with C++ classes, to explore how the major kinds of C++ language features associated with them interact (especially how behavior is or is not polymorphic under different situations), and to give you experience using those concepts within the Visual C++ environment.

In this studio you will again work in self-selected small groups of 2 or 3 people. As before, students who are more familiar with the material are encouraged to help those for whom it is less familiar. Asking questions of your professor and teaching assistants (as well as of each other) during the studio sessions is highly encouraged as well.

Please record your answers as you work through the following exercises. After you have finished please post your answers to the required exercises, and to any of the enrichment exercises you completed, to the course message board as a reply to my posting titled “Inheritance Studio”. The enrichment exercises are optional but are a good way to dig into the material a little deeper, especially if you breeze through the required ones. **Please make sure that as you work through these exercises that each member of your team has a chance to participate actively – one way is to take turns coding, looking up details, debugging, etc., and please also refer to the slides and the posted code examples as you work.**

### PART I: REQUIRED EXERCISES

1. Find your team members in the studio area, sit down at/around and log in to one of the Windows machines. Open up Visual Studio, and create a new Visual C++ Win32 Console Application project for this studio. Change the signature of the main function in the source file that Visual C++ generated to match the one that was specified in the lab assignments and in the lecture slides. Write down the names of the team members who are present (please catch up anyone arriving late on the work, and also add their name) as the answer to this exercise.
2. Add a new header file and source file to your project, in which you will (respectively) declare and define classes throughout this studio. In the header file, declare a class that has a public default constructor and public destructor. In the source file, define those methods so that each one prints out a statement indicating the class name and the method name, to `cout`. In the header file declare another class (the *derived class*) that is derived from the other class (which is the *base class*) via public inheritance. In the source file define the classes’ methods so that each of them also prints out its appropriate class name and method name. In your main source file, declare an object of each of the class types. Build and run your program, and explain **why** the program produced the output it did, as the answer to this exercise. **Hint:** you should see more constructors and destructors being called than just one of each per object.

3. Add another public method to each of the classes (with the same name in each) that takes no parameters and has a void return type. In the definition of that method (which should be in your source file) have it print out a statement indicating the class to which it belongs and its method name. Extend your main function from exercise 2 by adding two references to the base class type and one reference to the derived class type. Initialize one of the base class references using the base class object from exercise 2, and initialize the other base class reference and the derived class reference using the derived class object from exercise 2. Call the public method you added for this exercise using both of the objects and all three of the references. Put output statements before each of the calls to the function, so you can tell which of them produced which output.

Build and run the program, and record the output your program produced. In both of the classes put the keyword **virtual** before the declaration of the method you added above. Build and run the program again, and as the answer to this exercise, explain (a) **when**, (b) **how**, and (c) **why** the program's output differed when you used the keyword **virtual** before the method, vs. when you didn't.

4. Add two pointers to the base class type and one pointer to the derived class type to your main function. Initialize one of the base class pointers using the address of the base class object from exercise 2, and initialize the other base class pointer and the derived class pointer using the address of the derived class object from exercise 2. Repeat exercise 3 using the pointers. As the answer to this exercise explain whether the virtual vs. non-virtual method calls using the pointers act like the calls made using the objects or like the calls made using the references, and why that is so.

5. Replace the contents of your main function from the previous exercises with declarations of an object of the base type, an object of the derived type, two pointers to the base class type and one pointer to the derived class type. Initialize the pointers using calls to the **new** operator which creates a new instance of the base class type (for one of the base class pointers) or of the derived class type (for the other base class pointer and the derived class pointer). After all three of the pointers are initialized, destroy the newly created objects using the **delete** operator on each pointer. For example, **Base \* p = new Base;** etc. and then **delete p;** etc. As the answer to this exercise, compare the program's behavior (as indicated by its constructor and destructor outputs) when you declare both of the class destructors with vs. without the keyword **virtual** before them. **Hint:** you should again see more constructors (and especially with a virtual destructor, more destructors) than calls to **new** and **delete**.

6. With the method you added in exercise 3 and the destructor both made **virtual** in both the base and derived classes, try passing base class and derived class objects and pointers (and for base class pointers try passing pointers initialized with the addresses of base class and derived class objects) into method calls by reference vs. by value and inside those functions calling the method you added in exercise 3. As the answer to this exercise, please answer the following questions: (a) when does the behavior of the object inside the function match the original object being passed (or the object to which the pointer being passed points)?; (b) when does it act like an object of the derived class type has been "sliced" down to being an object of the base class type (this is sometimes called the *class slicing* problem)?

PART II: ENRICHMENT EXERCISES (optional, feel free to skip some and do ones that interest you).

7. In your main function, add try/catch logic, with separate catch blocks for objects of the derived and base class types. Try throwing and catching objects of the derived and base types, and in each catch block call the virtual method you added in exercise 3 on the object you caught. As the answer to this exercise, please answer the following questions: (a) what happens if you catch the exceptions by value vs. by reference? (b) what happens if you reverse the order of the catch blocks?

8. What happens to the behavior of the program when you change the relationship between the base and derived classes from public inheritance to protected or private inheritance? As the answer to this exercise, please answer the following questions based on changing the kind of inheritance: (a) does the behavior of the virtual method you added in exercise 4 change when it's called in the different cases you examined in the earlier exercises? (b) does the behavior of the virtual destructor change when you repeat exercise 7? (c) using the scoping operator to specify the specific class version of the method, which methods of which classes can or cannot be accessed on derived class objects in main?

9. Using the class relationships used in exercises 2 through 7 (public inheritance), add a third class that has an object of the base class and an object of the derived class as members. Add a default constructor and virtual destructor that print out their class and method names as in the previous exercises, and use the **new** and **delete** operators to create and destroy an instance of that third class in your main function. As the answer to this exercise, please describe the order of construction and destruction that occurs, and explain why it happens that way.

10. What happens if you declare one of the base class methods pure virtual (also known as making it abstract)? Can you still declare instances of that class? Or, can you only declare instances of derived classes that override the method? Can you declare instances of classes that have base class objects as members? As the answer to this exercise, please respond to these questions, and please explain why what you saw is happening.