**Programming Exercises - Inheritance**

1. Create a new empty Win32 Console Application. Create a class called ***Base*** by adding a new header file(.h) and source file(.cpp) to your project, in which you will (respectively) declare and define classes(You can use the Visual studio 'New Class' to do this). 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**. Create another class called ***Derived*** (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’ constructor and destructor methods so that each of them also prints out its appropriate class name and method name. In another source file create a main() function and 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.
2. 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 1 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 1, and initialize the other base class reference and the derived class reference using the derived class object from exercise 1. 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, versus when you didn’t.

1. Within your main function add two pointers to the base class type and one pointer to the derived class type. Initialize one of the base class pointers using the address of the base class object from exercise 1, and initialize the other base class pointer and the derived class pointer using the address of the derived class object from exercise 1. Repeat exercise 2 using the pointers. For 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.
2. 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 behaviour (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**.
3. With the method you added in exercise 2 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 2. As the answer to this exercise, please answer the following questions: (a) when does the behaviour 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)?