1. Create **Car.cpp** so that it also inherits from a class called **Vehicle**, placing appropriate member functions in **Vehicle** (that is, make up some member functions). Add a non-default constructor to **Vehicle**, which you must call inside **Car**’s constructor.

2. Create two classes, **A** and **B,** with default constructors that announce themselves. Inherit a new class called **C** from **A**, and create a member object of **B** in **C**, but do not create a constructor for **C**. Create an object of class **C** and observe the results.

3. Create a three-level hierarchy of classes with default constructors, along with destructors, both of which announce them-selves to **cout**. Verify that for an object of the most derived type, all three constructors and destructors are automatically called. Explain the order in which the calls are made.

4. Write a class containing a **long** and use the pseudo constructor call syntax in the constructor to initialize the **long**.

5. Create two classes called **Traveler** and **Pager** without default constructors, but with constructors that take an argument of type **string**, which they simply copy to an internal **string** variable. For each class, write the correct copy-constructor and assignment operator. Now inherit a class **BusinessTraveler** from **Traveler** and give it a member object of type **Pager**. Write the correct default constructor, a constructor that takes a **string** argument, a copy-constructor, and an assignment operator. 15. Create a class with two **static** member functions. Inherit from this class and redefine one of the member functions. Show that the other is hidden in the derived class.

6. Use **private** and **protected** inheritance to create two new classes from a base class. Then attempt to upcast objects of the derived class to the base class. Explain what happens.

7. Create **Protected.cpp**, add a member function in **Derived** that calls the **protectedBase** member **read( )**.

8. Change **Protected.cpp** so that **Derived** is using **protected** inheritance. See if you can call **value( )** for a **Derived** object.

9. Create a class called **SpaceShip** with a **fly( )** method. Inherit **Shuttle** from **SpaceShip** and add a **land( )** method. Create a new **Shuttle**, upcast by pointer or reference to a **SpaceShip**, and try to call the **land( )** method. Explain the results.

10. Create a class **Rock** with a default constructor, a copyconstructor, an assignment operator, and a destructor, all of which announce to **cout** that they’ve been called. In **main( )**, create a **vector<Rock>** (that is, hold **Rock** objects by value) and add some **Rock**s. Run the program and explain the output you get. Note whether the destructors are called for the **Rock** objects in the **vector**. Now repeat the exercise with a **vector<Rock\*>**. Is it possible to create a **vector<Rock&>**?

11. Start with a base class **Subject** and give it three functions: **f( )**, **g( )**, and **h( )**. Now inherit a class **Proxy** and two classes **Implementation1** and **Implementation2** from **Subject**. **Proxy** should contain a pointer to a **Subject**, and all the member functions for **Proxy** should just turn around and make the same calls through the **Subject** pointer. The **Proxy** constructor takes a pointer to a **Subject** that is installed in the **Proxy**(usually by the constructor). In **main( )**, create two different **Proxy** objects that use the two different implementations. Now modify **Proxy** so that you can dynamically change implementations.