# *C++ Primer Plus,* $5^{th}$ *Edition* by Stephen Prata
## Chapter 13: Class Inheritance
## Review Questions

1. What does a derived class inherit from a base class?

   *We inherit all private, public, and protected class member functions and variables. However, the derived class does not inherit any constructors, destructors, assignment operators, and friend functions of the base class*

2. What doesn't a derived class inherit from a base class?

   *See 1*

3. Suppose the return type for the `baseDMA::operator=()` function were defined as `void` instead of `baseDMA &`. What effect, if any, would that have? What if the return type were `baseDMA` instead of `baseDMA &`?

   *If the return type were void, then statements such as*

   ```
   baseDMA b1;
   baseDMA b2;
   baseDMA b3;
   b1 = b2 = b3;
   ```

   would not be possible. Since the assignment operator operates from right to left, the last statement is setting the value of `b1` to `b2 = b3`, which has a return value of void. If the return type was an object rather than a reference to an object, then the copy constructor would be invoked to create the object returned. This makes the program run a bit slower, but it is otherwise harmless unless `baseDMA` uses dynamic memory allocation and a copy constructor is not explicitly defined to do deep copying. However, the user must keep in mind that the object returned is not the object used to invoke the overloaded equality operator, so passing the returned object as a reference in a function which modifies it will leave the original object unchanged.

4. In what order are class constructors and class destructors called when a derived-class object is created and deleted?

   *When the constructor is invoked for the derived class, the base class constructor is is called first, and then the derived class constructor is called. When the destructor is invoked for the derived class, the derived class destructor is called first and then the base class destructor is called.*

5. If a derived class doesn't add any data members to the base class, does the derived class require constructors?

*If the defined class does not necessarily require an explicit constructor, because a default is provided in the event that one is not defined. However, the default derived class constructor is a do nothing constructor that calls the default constructor for the base class. If this is not what the user wants, it is best to specify a constructor for the derived class explicitly, because it gives the user control as to which constructor is called for the base class as well, even if the derived class constructor is do nothing.*

6. Suppose a base class and a derived class both define a method with the same name and a derived-class object invokes the method. What method is called?

   *There are two scenarios:*

   (1) *The methods are not virtual:*
   *If a pointer of type ¡base class¿ is pointing to the derived class, invoking the method through the pointer will invoke the base class method. If a reference of type ¡base class¿ is assigned to the derived class, the invoking method through the reference will invoke the base class method.*

   (2) *The methods are vitual If a pointer of type ¡base class¿ is pointing to the derived class, invoking the method through the pointer will invoke the derived class method. If a reference of type ¡base class¿ is assigned to the derived class, the invoking method through the reference will invoke the derived class method.*

7. When should a derived class define an assignment operator?

   *An assignment operator should be explicitly defined in the case that the derived class object has a dynamically allocated object that was not inherited from the base class. The overloaded operator should do deep copying. It is also important to note that in this case the copy constructor should be explicitly defined as well.*

8. Can you assign the address of an object of a derived class to a pointer to the base class? Can you assign the address of an object of a base class to a pointer to the derived class?

   *Assigning the address of a derived class to a pointer to the base class is called upcasting and it may be done without an explicit type cast. However, the opposite is called downcasting and can only be done with an explicit type cast.*

9. Can you assign an object of a derived class to an object of the base class? Can you assign an object of a base class to an object of the derived class?

   *Because of implicit upcasting, you can assign an object of a derived class to an object of the base class. Without an explicit type cast or conversion constructor, you cannot assign an object of the derived class to an object of the base class.*

10. Suppose you define a function that takes a reference to a base-class object as an argument. Why can this function also use a derived-class object as an argument?

    *Yes, this is permissible because upcasting is implicit. Since the derived class inherits member values and functions from the base class, any derived class methods will work when performed on the base class.*

11. Suppose you define a function that takes a base-class object as an argument (that is, the function passes a base-class object by value). Why can this function also use a derived-class object as an argument?

    *When a function's formal parameter is an object, the copy constructor is used to create a temporary object which is used in the body of the function. Because of implicit upcasting, the copy constructor for the base class object may take a derived class object as its actual argument without any problems.*

12. Why is it usually better to pass objects by reference than by value?

    *Passing by reference obviates the need to create a temporary object with the copy constructor and thus makes the program faster.*

13. Suppose `Corporation` is a base class and `PublicCorporation` is a derived class. Also suppose that each class defines a `head()` member function, that `ph` is a pointer to the `Corporation` type, and that `ph` is assigned the address of a `PublicCorporation` object. How is `ph->head()` interpreted if the base class defines `head()` as a

    a. Regular nonvirtual method

       *The program runs* `Corporation::head()`.

    b. Virtual method

       *The program runs* `PublicCorporation::head()`.

14. What's wrong, if anything, with the following code?

```
class Kitchen
{
private:
    double kit_sq_ft;
public:
    Kitchen() {kit_sq_ft = 0.0; }
    virtual double area() const { return kit_sq_ft * kit_sq_ft; }
};
class House : public Kitchen
{
private:
    double all_sq_ft;
public:
    House() {all_sq_ft += kit_sq_ft;}
    double area(const char *s) const { cout << s; return all_sq_ft; }
};
```

*Several things are wrong with this code.*

(1) *We have a base class and a derived class, but the derived class relationship with the base class does not follow an* ***is-a*** *relationship.*

(2) *The variable* `all_sq_ft` *is never initialized, so we are in store for unpleasant surprises when we attempt to print or use the value.*

(3) *There is no point to making* `area()` *a virtual function if the* `area()` *function in the derived class has a different function signiature.*

(4) *We need to include the* `iostream` *library and include a using declaration so that we may use* `cout` *in the* `area()` *function of the* `House` *object.*

(5) *The base class should have a virtual destructor, which it does not.*