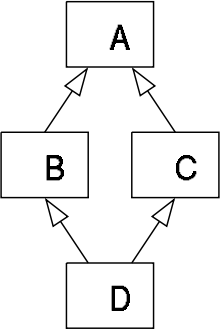
**Solving the Diamond Problem with Virtual Inheritance**

**Multiple inheritance** in C++ is a powerful, but tricky tool, that often leads to problems if not used carefully. This article will teach you how to use virtual inheritance to solve a common problem programmers run into, the diamond problem.

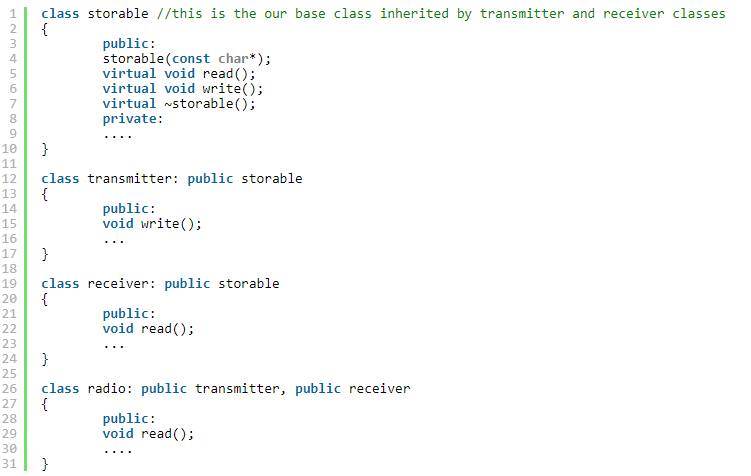
**The diamond problem**



The "diamond problem" is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C. If there is a method in A that B and C have [overridden](https://en.wikipedia.org/wiki/Method_overriding_(programming)), and D does not override it, then which version of the method does D inherit: that of B, or that of C?

A classical illustration of this is given by Bjarne Stroustrup (the creator of C++) in the following example:

**Sample code**



**Problem**

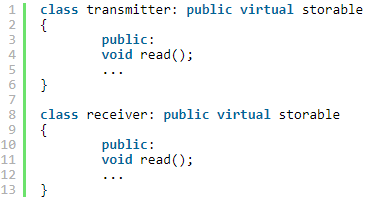
1. Since both **transmitter** and **receiver** classes are using the method **write()** from the base class, when calling the method **write()** from a **radio** object the call is ambiguous.
2. The compiler can't know which implementation of **write()** to use, the one from the **transmitter** class or the one from the **receiver** class.

**What happened?**

1. In memory, inheritance simply puts the implementation of two objects one after another.
2. But **radio** is both a **transmitter** and a **receiver**, so the **storable** class gets duplicated inside the radio object.
3. Compile error: ‘request for member “**write**” is ambiguous’, because it can't figure out whether to call the method **write()** from **receiver** or from **transmitter**.

**Solution: Virtual Inheritance**

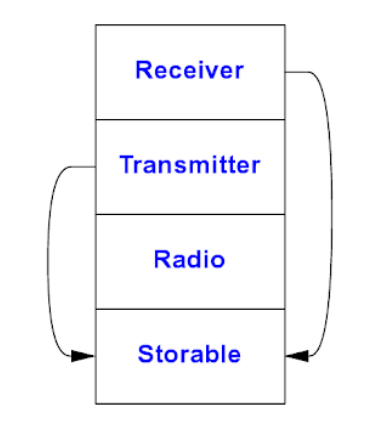
In order to prevent the compiler from giving an error we use the keyword **virtual** when we inherit from the base class **storable** in both derived classes:



1. When we use virtual inheritance, we are guaranteed to get only a single instance of the common base class.
2. In other words, the **radio** class will have only a single instance of the **storable** class, shared by both the **transmitter** and **receiver** classes

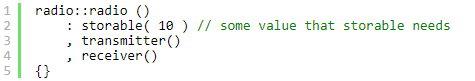
**Memory Layout in Virtual Inheritance**

When a **radio** object is constructed, it creates one **storable** instance, a **transmitter** instance and a **receiver** instance:



**Constructors behind Virtual Inheritance**

1. Because there is only a single instance of a virtual base class that is shared by multiple classes that inherit from it, the constructor for a virtual base class is not called by the class that inherits from it (which is how constructors are called, when each class has its own copy of its parent class)
   * Otherwise, that would mean the constructor would run multiple times.
2. Instead, the constructor is called by the constructor of the concrete class. In the example above, the class **radio** directly calls the constructor for **storable**.
   * If you need to pass any arguments to the **storable** constructor, you would do so using [an initialization list](https://www.cprogramming.com/tutorial/initialization-lists-c++.html), as usual:

****

1. One thing to be aware of is that if either **transmitter** or **receiver** attempted to invoke the **storable** constructor in their initialization lists, that call will be completely skipped when constructing a radio object!
   * Be careful, as this could cause a subtle bug!

Some more detailed rules for the constructor calls:

1. (**Constructors**) The constructors for virtual base classes are always called before the constructors for non-virtual base classes.
   * This ensures that a class inheriting from a virtual base class can be sure the virtual base class is safe to use inside the inheriting class's constructor.
2. (**Destructors**) The destructor order in a class hierarchy with a virtual base class follows the same rules as the rest of C++: the destructors run in the opposite order of the constructors.
   * In other words, the virtual base class will be the last object destroyed, because it is the first object that is fully constructed.

**References:**

1. <https://www.cprogramming.com/tutorial/virtual_inheritance.html> (by Andrei Milea)
2. The C++ Programming Language, 4th Edition (by Bjarne Stroustrup)
   * Section 21.3.4
   * Section 21.3.5