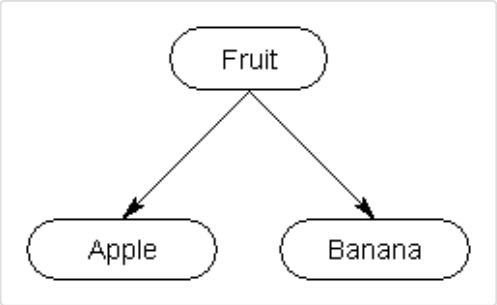
11.1 — Introduction to inheritance

BY ALEX ON DECEMBER 19TH, 2007 | LAST MODIFIED BY ALEX ON JANUARY 13TH, 2016

In the lesson on **composition**, you learned how to construct complex classes by combining simpler classes. Composition is perfect for building new objects that have a *has-a* relationship with their subobjects. However, composition (and aggregation) is just one of the two major ways that C++ lets you construct complex classes. The second way is through inheritance.

Unlike composition, which involves creating new objects by combining and connecting other objects, **inheritance** involves creating new objects by directly acquiring the attributes and behaviors of other objects and then extending or specializing them. Like composition, inheritance is everywhere in real life. You inherited your parents genes, and acquired physical attributes from both of them. Technological products (computers, cell phones, etc…) often inherit features from their predecessors. C++ inherited many features from C, the language upon which it is based, and C itself inherited many of its features from the programming languages that came before it.

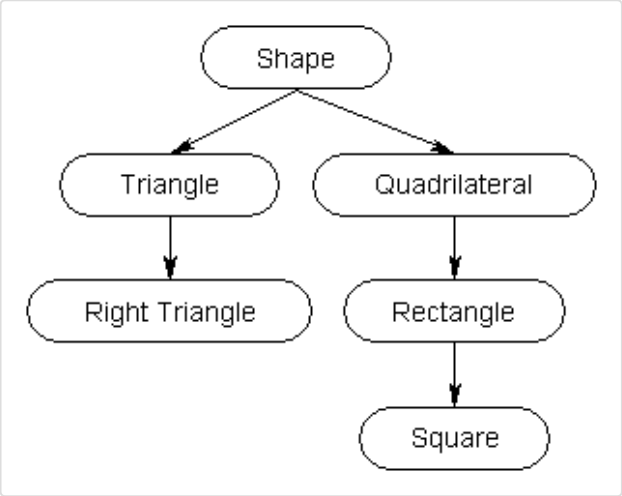
Consider an apple and a banana. Although an apple and a banana are different fruits, both have in common that they *are* fruits. Because apples and bananas *are* fruits, anything that is true of fruits is also true of apples and bananas. For example, all fruits have a name, a flavor, and are tangible objects. Thus, apples and bananas also have a name, a flavor, and are tangible objects. Apples and bananas inherit these properties from the concept of fruit because they *are* fruit. Apples and bananas then define some of these properties in different ways (apples and bananas have different flavors), which is what makes them distinct from each other.



The object being inherited from is called the **parent** or **base**, and the object doing the inheriting is called the **child** or **derived** object. In the above picture, “fruit” is the parent, and both “apple” and “banana” are children. Unlike in composition, where each object has a *has-a* relationship with its subobjects, in inheritance, each child has an *is-a* relationship with its parent. An apple *is-a* fruit. A triangle *is-a* shape. Red *is-a* color.

By default, the children receive all of the properties of the parents. However, the children are then free to define or redefine inherited properties (bananas have that unique banana flavor), add new properties (eg. bananas add the property of being “starchy”, which is not true of many other fruits), or even hide properties.

It is possible to define entire hierarchies of objects via inheritance. For example, a square is a rectangle, which is a quadrilateral, which is a shape. A right triangle is a triangle, which is also a shape.



Why the need for inheritance in C++?

One of the fundamental ideas behind object-oriented programming is that code should be reusable. However, existing code often does not do EXACTLY what you need it to. For example, what if you have a triangle and you need a square? In this case, we are presented with a number of choices on how to proceed, all of which have various benefits and downsides.

Perhaps the most obvious way to proceed is to change the existing code to do what you want. However, if we do this, we will no longer be able to use it for its original purpose, so this is rarely a good idea.

A slightly better idea is to make a copy of some or all of the existing code and change it to do what we want. However, this has several major downsides. First, although copy-and-paste seems simple enough, it's actually quite dangerous. A single omitted or misplaced line can cause the program to work incorrectly and can take days to find in a complex program. Renaming a class via search-and-replace can also be dangerous if you inadvertently replace something you didn't mean to. Second, to rewrite the code to make it do what you want, you need to have an intimate understanding what it does. This can be difficult when the code is complex and not adequately documented. Third, and perhaps most relevant, this generally involves duplicating of existing functionality, which causes a maintenance problem. Improvements or bug fixes have to be added to multiple copies of functions that do essentially the same thing, which wastes programmer time. And that's assuming the programmer realizes multiple copies even exist! If not, some copies may not get the improvements or bug fixes.

Inheritance solves most of these problems in an efficient way. Instead of manually copying and modifying every bit of code your program needs, inheritance allows you directly reuse existing code that meets your needs. You only need to add new features, redefine existing features that do not meet your needs, or hide features you do not want. This is typically much less work (as you are only defining what has changed compared to the base, rather than redefining everything), and safer too. Furthermore, any changes made to the base code automatically get propagated to the inherited code. This means it is possible to change one piece of code (eg. to apply a bug fix) and all derived objects will automatically be updated.

Inheritance does have a couple of potential downsides, but we will cover those in future lessons.

 **11.2 -- Basic inheritance in C++**

 **Index**

 **10.4 -- Container classes**