

## C++ Tip 0002 — Ordering of public, private, and protected Members

© 2009 Rex Jaeschke. All rights reserved.

**Issue:** I see several different styles with respect to ordering of members by access specifier groups. Does the ordering really matter?

**Response:** There are two main approaches to this: put all the private members last, and put all the private members first.

The argument for the first approach is that the private members are part of the “hidden” implementation details of the class. So, when someone is reading the class header, at least from an overview perspective, he/she should be oblivious to how the class is implemented. (Certainly, having the public interface specified all in one place makes sense; however, that has no bearing on the ordering of access specifier groups, just in grouping like-access specified members.) The second approach is favored by those who want to see all the details of class implementation when they first load the header into an editor.

While it may be uncommon in C++ code to see seemingly arbitrary orderings of members having different access specifiers, it is not uncommon in languages like Java and C#, where the access specifier *must* be written on *each* member declaration. In those languages, many programmers simply seem to declare the members as they think of them rather than following any ordering pattern

Occasionally, the “all private members first” approach can break down. For example,

```
class X
{
public:
    enum Color {red, blue, green};
private:
    Color c;
    // ...
};
```

Here, the declaration of `c` requires that the definition of the enum type `Color` be in scope, and that is only possible if that definition precedes the declaration of `c`.

Some programmers who use the “all private members first” first also write the `private` access specifier explicitly even though doing so is redundant.

One interesting case is a class containing a series of bit fields, some of which are public, some private, and some protected.

Within a given access specifier group, the address of successive non-static data members will be higher than their predecessor in that group, with alignment holes permitted between data members. However, the C++ Standard (2003), §9.2, “Class members”, p. 157, states, “The order of allocation of non-static data members separated by an *access-specifier* is unspecified (11.1).”

Simply stated, the ordering of members is, for the most part, stylistic.