*Document number:* **P0562R0**
*Date:* 2017-02-05
*Audience:* Evolution Working Group
*Reply to:* **Alan Talbot**
cpp@alantalbot.com

# Initialization List Symmetry

One of the requirements of writing solid C++ is the maintenance of initialization lists in constructors. The list must match the declarations in the class body both in content and order. Neither is enforced, but the hazards of omitting an entry or getting them out of order are well known.[1] This requirement is necessary given the flexibility and guarantees of the language, but it *is* rather tedious.

The syntax of initialization lists makes formatting them for maximum readability and maintainability something of a quandary. I have tried several different formats, but have found that there is no perfect answer. My current style for very short lists is usually this:

```
foo::foo(int x, int y) : a(x), b(y)
{...}
```

but for any non-trivial class I use this:

```
foo::foo(int x, int y, int z) :
   a(x),
   b(y),
   c(z)
{...}
```

I find this arrangement makes it easier to add to, remove from, and reorder the initializer list. Putting the colon on the first line followed by each initializer on its own line is almost ideal. The only problem is the last line. Every time a change involves the last line, I have to delete and add a comma somewhere. That may not seem like much work, but it adds up over time and it's easy to forget, which means a compile-time error which takes even more time.

The solution is very simple: allow a redundant trailing comma, exactly as is allowed in **enum** definitions, and for the same reason:

```
foo::foo(int x, int y, int z) :
   a(x),
   b(y),
   c(z),
{...}
```

This small change is entirely cosmetic and not very exciting—it doesn't change the abilities of the language—but I believe it will save the millions of programmers who use C++ a noticeable amount of time and energy over the years. And efficiency, at all levels, is part of what C++ is all about.

## Proposed Wording

[This wording is a first approximation. I am seeking guidance to refine it.]

**12.6.2 Initializing bases and members** **[class.base.init]**

¶1

In the definition of a constructor for a class, initializers for direct and virtual base subobjects and non-static data members can be specified by a ctor-initializer, which has the form

*ctor-initializer:*
*: mem-initializer-list*
*: mem-initializer-list ,*

*mem-initializer-list:*
*mem-initializer ...opt*
*mem-initializer-list , mem-initializer ...opt*

*mem-initializer:*
*mem-initializer-id ( expression-listopt )*
*mem-initializer-id braced-init-list*

*mem-initializer-id:*
*class-or-decltype*
*identifier*

**A.10 Special member functions** **[gram.special]**

*ctor-initializer:*
*: mem-initializer-list*
*: mem-initializer-list ,*

## Notes

1. For example: Scott Meyers. *Effective C++ Third Edition,* p. 28-29. Pearson Education, 2005.

## Acknowledgements

Thanks to Daveed Vandevoorde for confirming that the syntax is possible.