Note that this topic is deemed an annoyance, rather than a pitfall, because it affects only code newly written for C++11 or later using the new forms of initialization syntax, so it does not break existing C++03 code recompiled with a more modern language dialect. However, also note that many containers and other types in the C++ Standard Library inherited such a design and have not been refactored into multiple constructors, although some such refactoring occurs in later versions of the Standard.

Obfuscation due to opaque use of braced list initializers

Use of braced initializers for function arguments, omitting any hint of the expected object type at the call site, requires deep familiarity with functions being called to understand the actual types of arguments being initialized, especially when overload resolution must disambiguate several viable candidates. Such usage might produce more fragile code as further overloads are added, silently changing the type initialized by the braced list as a different function wins overload resolution. Such code is also much harder for a subsequent maintainer, or casual code reader, to understand:

```
struct C
{
    C(int, int) { }
};

int test(C, long) { return 0; }

int main()
{
    int a = test({1, 2}, 3);
    return a;
}
```

This program compiles and runs, returning the intended result. However, consider how the behavior changes if we add a second overload during subsequent maintenance:

```
struct C
{
    C(int, int) { }
};

int test(C, long) { return 0; }

struct A  // additional aggregate class
{
    int x;
    int y;
};
```