`std::initializer_list` constructor suppresses the implicitly declared default

The declaration of a constructor with an *initializer-list* argument will suppress the implicit declaration of a default constructor, as one would expect. Without a default constructor, the `std::initializer_list` constructor will be called when the object is initialized from an empty list but not in other circumstances where a default constructor might be called. If such a type is then used as the type of a subobject, it would result in an implicitly declared default constructor of the outer object being deleted. These rules can make initialization from a pair of empty braces a bit counterintuitive:

```
#include <cassert>  // standard C assert macro
#include <vector>   // std::vector

struct X
{
    std::vector<int> d_v;

    X(std::initializer_list<int> il) : d_v(il) {}
};

struct Y
{
    long long d_data;

    Y() : d_data(-1) {}
    Y(std::initializer_list<int> il) : d_data(il.size()) {}
};

struct Z1 : X { };

struct Z2 : X
{
    Z2() = default;       // BAD IDEA, implicitly deleted
};

struct Z3 : X
{
    using X::X;
};

struct Z4
{
    X data;

    Z4() = default;
};
```