```
int  b[] = { 0, i, j };  // Error, cannot narrow j from long to int

struct S { int a; };
S s1 = { 0  };  // OK
S s2 = { i  };  // OK
S s3 = { 0L };  // OK, 0L is an integer constant expression.
S s4 = { j  };  // Error, narrowing
```

In addition, the rules for value initialization now state that members without a specific initializer value in the braced list are "as-if" copy initialized from `{}`. These rules will result in an error when initializing a member that has an **`explicit`** default constructor. Furthermore, from C++14 onward, if such a member has a default member initializer, then that member is initialized from the default member initializer; see Section 1.2."Aggregate Init '14" on page 138. Note that if the member is of reference type and no initializer is provided, the initialization is ill formed.

Regardless of whether the aggregate itself is initialized using a copy initialization or direct initialization, the members of the aggregate will be copy initialized from the corresponding initializer:

```
struct E { };                          // empty type
struct AE { int x; E y; E z; };        // aggregate comprising several empty objects
struct S { explicit S(int = 0) {} };   // class with explicit default constructor
struct AS{ int x; S y; S z; };         // aggregate comprising several S objects

AE aed;                    // OK
AE ae0 = {};               // OK
AE ae1 = { 0 };            // OK
AE ae2 = { 0, {} };        // OK
AE ae3 = { 0, {}, {} };    // OK

AS asd;                    // OK
AS as0 = {};               // OK in 03; Error in 11 calling explicit ctor for S
AS as1 = { 0 };            // OK in 03; Error in 11 calling explicit ctor for S
AS as2 = { 0, S() };       // OK in 03; Error in 11 calling explicit ctor for S
AS as3 = { 0, S(), S() };  // OK, all the aggregate's members have an initializer.
```

To better support generalizing the syntax of brace initialization in a style similar to aggregate initialization, an aggregate can be initialized from an object of the same type through *aggregate* initialization in C++11 as well as through *direct* initialization per C++03:

```
S x{};      // OK, value initialization
S y = {x};  // OK in C++11; copy initialization via aggregate-initialization syntax
```

Otherwise, initialization of aggregates in C++11 is the same where it would have a meaning in C++03 and is correspondingly extended into new places where braced initialization is permitted, as documented in the following subsections.