

Section 2.1 C++11

Braced Init

function1 in the previous code example, so these functions do not work for aggregates prior to C++20.

Uniform member initialization in generic code

With the addition of general **braced initialization** to C++11, class authors should consider whether constructors should use **direct initialization** or **direct list initialization** to initialize their bases and **members**. Note that since **copy initialization** and **copy list initialization** are not options, whether or not the constructor for a given base or **member** is **explicit** will never be a concern.

Prior to C++11, writing code that initialized **aggregate** subobjects, including arrays, with a set of data in the constructor’s **member initializer list** was not really possible. We could only **default-initialize**, **value-initialize**, or **direct-initialize** from another **aggregate** of the same type.

Starting with C++11, we are able to initialize **aggregate members** with a list of values, using **aggregate initialization** in place of **direct list initialization** for **members** that are **aggregates**:

```

struct S
{
    int i;
    std::string str;
};

class C
{
    int j;
    int a[3];
    S s;

public:
    C(int x, int y, int z, int n, const std::string& t)
      : j(0)
      , a{ x, y, z } // ill formed in C++03, OK in C++11
      , s{ n, t }   // ill formed in C++03, OK in C++11
      {
      }
};

```

Note that as the initializer for `C.j` shows in the code example above, there is no requirement to consistently use either braces or parentheses for all **member** initializers.

As with the case of **factory functions**, the author of a generic class must make a choice for constructors between adding support for initializing **aggregates** versus **narrowing conversion** being **ill formed**. As mentioned earlier, since **member** initialization supports only **direct list initialization**, there is never a concern regarding **explicit** conversions in this context: