```
#include <type_traits>  // std::is_trivial

struct S0 { int d_i;     };
struct S1 { int d_i = 0; };
struct S2 { int d_i; S2() : d_i(0) { } };

static_assert(std::is_trivial<S0>::value, "");
static_assert(!std::is_trivial<S1>::value, "");
static_assert(!std::is_trivial<S2>::value, "");
```

### Loss of aggregate status

In C++11, classes using default member initializers are not considered **aggregates**, and therefore **aggregate initialization** can't be used. Fortunately, this restriction has been lifted in C++14; see Section 1.2."Aggregate Init '14" on page 138:

```
struct ThreadPoolConfiguration
{
    int  d_numThreads         = 8;     // number of worker threads
    bool d_enableWorkStealing = true;  // enable work stealing
    int  d_taskSize           = 64;    // buffer size for an enqueued task
};

void f()
{
    ThreadPoolConfiguration tpc0;               // OK in C++11
    ThreadPoolConfiguration tpc1{16, true, 64}; // Error, in C++11; OK in C++14
}
```

### Default member initializer does not deduce array size

Default member initializers do not allow deduction of the size of an array member:

```
struct S
{
    char s[]{"Idle"};  // Error, must specify array size
};
```

The rationale is that there is no guarantee that the default member initializer will be used to initialize the member; hence, it cannot be a definitive source of information about the size of such a member in the object layout.