```
struct X { int i, j; double d[2]; };     X x = {};  // OK, X is an aggregate.
class  Y { int i; public: Y(); ~Y(); };  Y y = {};  // Error, Y isn't.
```

A *C++03* aggregate is an array, **class**, **struct**, or **union** having no **user-declared**[6] constructors, no **private** or **protected** non**static** data members, no base classes, and no **virtual** functions:

```
// Class declaration                    Is a C++03 aggregate?
class  A0  { };                      // Yes, empty class is an aggregate.
class  A1  { int x; };               // no, private data member
class  A2  { protected: int x; };    // no, protected data member
class  A3  { public: int x; };       // yes, public data
class  A4  { int f(); };             // yes, private nonvirtual function
class  A5  { static A1 x; };         // Yes, static members don't matter.
struct A6  { A6() { } };             // no, user-declared default ctor
struct A7  { A7(const A7&) { } };    // no, user-declared copy ctor
struct A8  { A8(int) { } };          // no, user-declared value ctor
struct A9  { ~A9(); };               // Yes, destructor can be declared.
struct A10 { A10& operator=(const A10&); };
                                     // yes, user-declared copy assignment allowed
struct A11 { int* x; };              // Yes, pointers are allowed in aggregates.
struct A12 : A0 { };                 // no, base class
struct A13 { virtual void f(); };    // no, virtual function
struct A14 { A1  x; };               // Yes, data members need not be aggregates.
struct A15 { A13 x; };               // Yes,   "     "      "   "   "     "

struct A16 { const int x; };         // yes, but must initialize const values
struct A17 { int& x; };              // yes,  "    "      "      references
union  A18 { int x; double y; };     // Yes, unions can be aggregates.
```

As the example types above illustrate, an aggregate may contain arbitrary public data members, private nonvirtual functions, and static members of any kind. Although an aggregate may not declare any constructors, it is permitted to declare a copy-assignment operator and a destructor. Importantly, an aggregate is permitted to contain elements that are themselves not of aggregate type. Hence, an array of any C++ type would itself be considered an aggregate:

```
#include <string>  // std::string
std::string a[10] = {};  // a is an aggregate.
```

[6]The C++03 term user-declared is replaced in C++11 by **user-provided** because a **special member function** that is explicitly declared and immediately **defaulted** (see Section 1.1."Defaulted Functions" on page 33) or **deleted** (see Section 1.1."Deleted Functions" on page 53) is considered user-declared and yet is not user-provided; hence, a class with, e.g., explicitly defaulted constructors can still be an aggregate in C++11.