```
union U1 { bool b = 0;  char c;       };  // OK, first member initialized
union U2 { bool b;      char c = 'A'; };  // OK, second   "        "
union U3 { bool b = 0;  char c = 'A'; };  // Error, multiple initialized

// unions having constexpr constructors
union U4 { bool b; char c;        constexpr U4() { } };  // Error, uninit
union U5 { bool b; char c = 'A';  constexpr U5() { } };              // OK
union U6 { bool b; char c;        constexpr U6() : c('A') { } };     // OK
union U7 { bool b; char c;        constexpr U7(bool v) : b(v) { } }; // OK

struct S                          // S is a literal type.
{
    U0 u0{};                      // value-initialized
    U1 u1; U2 u2; U5 u5; U6 u6;   // default-initialized
    U7 u7;                        // initialized in constructor
    constexpr S() : u7(true) { }  // OK, all members are initialized.
};

constexpr int test(S t) { return 0; }  // OK, confirms S is a literal type
```

The example code above illustrates various ways in which unions, e.g., `U0`–`U2` and `U5`–`U7`, can be used that allow them to be initialized by a **`constexpr`** constructor, e.g., `S()`. The existence of at least one non*copy*, non*move* **`constexpr`** constructor implies that the class, e.g., `S`, comprising these unions is a literal type, which we have confirmed using the C++11 interface test idiom; see *Identifying literal types* on page 282.

8. If the constructor *delegates* to another constructor in the same class (see Section 1.1. "Delegating Ctors" on page 46), that target constructor must be **`constexpr`**:

```
struct C0  // Only the default constructor is constexpr.
{
              C0(int)      { }  // OK, but not declared constexpr
    constexpr C0() : C0(0) { }  // Error, delegating to nonconstexpr ctor
};

struct C1  // Both default and value constructor are constexpr.
{
    constexpr C1(int)      { }  // OK, declared constexpr
    constexpr C1() : C1(0) { }  // OK, delegating to constexpr constructor
};
```

9. When initializing data members of a class (e.g., `S` below), any nonconstructor functions ~~needed for~~ implicitly convert~~ing~~ the type of the initializing expression (e.g., `V` in the code snippet below) to that of a data member (e.g., **`int`** or **`double`**) must also be **`constexpr`**: