```
namespace n           // enclosing namespace
{

class C { /*...*/ };  // arbitrary class definition

struct S
{
    constexpr S(bool) try { } catch (...) { }  // Error, function try block
              S(char) try { } catch (...) { }  // OK, not declared constexpr

    constexpr S(int)
    {
        ;                       // OK, null statement
        static_assert(1, "");   // OK, static_assert declaration
        typedef int Int;        // OK, simple typedef alias
        using Int = int;        // OK, simple using alias
        typedef enum {} E;      // Error, typedef used to define enum E
        using n::C;             // OK, using declaration
        using namespace n;      // OK, using directive
    }
};

}  // close namespace
```

2. All nonstatic data members and base-class subobjects of a class must be initialized by a **constexpr** constructor,[4] and the initializers themselves must be usable in a constant expression. Scalar members must be explicitly initialized in the member-initializer list or via a **default member initializer**, i.e., they cannot be left in an uninitialized state:

```
struct B  // constexpr constructible only from argument convertible to int
{
    B() { }
    constexpr B(int) { }  // constexpr constructor taking an int
};

struct C  //  constexpr default constructible
{
    constexpr C() { }  // constexpr default constructor
};

struct D1 : B  // public derivation
{
```

[4]The requirement that all members and base classes be initialized by a constructor that is explicitly declared **constexpr** is relaxed in C++20 provided that uninitialized **entities** are not accessed at compile time.