```
    assert(x < 100);          // Error, no runtime asserts
    int y = x;                // Error, no local variables
    if (x > 5) { return x; }  // Error, no if statements
    using S = struct { };     // Error, no aliases that define types
    return x += 3;            // Error, no compound assignment
}
```

The good news is that the aforementioned restrictions on the kinds of constructs that are permitted in **constexpr** function bodies are significantly relaxed as of C++14; see Section 2.2."**constexpr** Functions '14" on page 959.

Irrespective of the *kinds* of constructs that are allowed to appear in a **constexpr** function body, every invocation of a function, a constructor, or an implicit conversion operator in the **return** statement must itself be usable in at least one constant expression, which means the corresponding function *must*, at a minimum, be declared **constexpr**:

```
          int ga() { return 0; }  // nonconstexpr function returning 0
constexpr int gb() { return 0; }  //    constexpr function returning 0

struct S1a {          S1a() { } };  // nonconstexpr default constructor
struct S1b { constexpr S1b() { } };  //    constexpr default constructor

struct S2a { operator int() { return 5; } };         // nonconstexpr conversion
struct S2b { constexpr operator int() { return 5; } };  // constexpr conversion

constexpr int f1a() { return ga(); }  // Error, ga is not constexpr.
constexpr int f1b() { return gb(); }  // OK, gb is constexpr.

constexpr int f2a() { return S1a(), 5; }  // Error, S1a ctor is not constexpr.
constexpr int f2b() { return S1b(), 5; }  // OK, S1b ctor is constexpr.

constexpr int f3a() { return S2a(); } // Error, S2a conversion is not constexpr.
constexpr int f3b() { return S2b(); } // OK, S2b conversion is constexpr.
```

Note that non**constexpr** implicit conversions, as illustrated by `f3a` above, can also result from a non**constexpr**, non**explicit** constructor that accepts a single argument.

Constraints specific to constructors

In addition to the general restrictions on a **constexpr** function's body (see *Restrictions on **constexpr** function bodies (C++11 only)* on page 268) and its allowed parameter and return types (see ***constexpr**-function parameter and return types* on page 277), several additional requirements are specific to constructors.

1. The body of a **constexpr** constructor is restricted in the same way as any other **constexpr** function, except that the **return** statement is disallowed. Hence, the body of a **constexpr** constructor must be essentially empty with few exceptions: