Notice that declaring a member function, such as `setY` in the code example above, to be **constexpr** implicitly qualifies the member function as being **const**, thereby making it an error for any **constexpr** member function to attempt to modify its own object's data members. The inevitable corollary is that any appropriate implementation of *copy* or *move* assignment cannot be declared **constexpr** in C++11 but can be as of C++14.

Finally, **constexpr** member functions cannot be **virtual**[3] but can co-exist in the same class with other member functions that *are* virtual.

Restrictions on `constexpr` function bodies (C++11 only)

The list of C++ programming features permitted in the bodies of **constexpr** functions for C++11 is small and reflective of the nascent state of this feature when it was first standardized. To begin, the body of a **constexpr** function is not permitted to be a **function-try-block**:

```
          int g1()     { return 0; }               // OK
constexpr int g2()     { return 0; }               // OK, no try block
          int g3() try { return 0; } catch(...) {}  // OK, not constexpr
constexpr int g4() try { return 0; } catch(...) {}  // Error, not allowed
```

C++11 **constexpr** functions that are not *deleted* or *defaulted* (see Section 1.1."Deleted Functions" on page 53 and Section 1.1."Defaulted Functions" on page 33, respectively) may consist of only **null statements**, static assertions (see Section 1.1."**static_assert**" on page 115), **using declarations**, **using directives**, and **typedef** and alias declarations (see Section 1.1."**using** Aliases" on page 133) that do not define a class or enumeration. Other than constructors, the body of a **constexpr** function must include exactly one **return** statement. A **constexpr** constructor may have a member-initializer list ~~but no other additional statements~~, ~~but see *Constraints specific to constructors* on page 269.~~ Use of the **ternary operator**, **comma operator**, and recursion is allowed:

```
constexpr int f(int x)
{
    ;                                      // OK, null statement
    static_assert(sizeof(int) == 4, "");   // OK, static assertion
    using MyInt = int;                     // OK, type alias
    return x > 5 ? x : f(x + 2), f(x + 1); // OK, ternary, comma, and recursion
}
```

Many familiar programming constructs such as runtime assertions, local variables, **if** statements, modifications of function parameters, and ~~**using** directives~~ that define a type are, however, *not* permitted in C++11:

```
#include <cassert>  // standard C assert macro
constexpr int g(int x)
{
```

[3]C++20 allows **constexpr** member functions to be **virtual** (**dimov18**).