Annoyances

Deleting a function declares it

It should come as no surprise that when we declare a **free function** followed by `= delete`, we *are* in fact *declaring* it. For example, consider the pair of overloads of functions `f` declared taking a `char` and `int`, respectively:

```
int f(char);          // (1) accessible declaration of f taking a char
int f(int) = delete;  // (2) inaccessible declaration of f taking an int

int x = f('a');       // OK, exact match for (1) f(char), which is accessible
int y = f(123);       // Error, exact match for (2) f(int), which is deleted
```

Both functions above must be *declared* so that both of them can participate in overload resolution; it is only after the inaccessible overload is selected that it will be reported as a compile-time error.

When it comes to deleting certain special member functions of a class (or class template), however, what might seem like a tiny bit of extra, self-documenting code can have subtle, unintended consequences as evidenced below. Let's begin by considering an empty `struct`, `S0`:

```
struct S0 { };  // The default constructor is declared implicitly.

S0 x0;  // OK, invokes the implicitly generated default constructor
```

As `S0` defines not constructors, destructors, or assignment operators, the compiler will generate (declare and **define**), for `S0`, all *six* of the special member functions available as of C++11; see Section 1.1."Defaulted Functions" on page 33.

Next, suppose we create a second `struct`, `S1`, that differs from `S0` only in that `S1` declares a *value* constructor taking an `int`:

```
struct S1  // Implicit declaration of the default constructor is suppressed.
{
    S1(int);  // explicit declaration of value constructor
};

S1 y1(5);  // OK, invokes the explicitly declared value constructor
S1 x1;     // Error, no declaration for default constructor S1::S1()
```

By explicitly declaring a *value* constructor (or any other constructor for that matter), we automatically suppress the implicit declaration of the default constructor for `S1`. If suppressing the default destructor is *not* our intention, we can always reinstate it via an explicit declaration followed by `= default;` (see Section 1.1."Defaulted Functions" on page 33).