Further Reading

- For yet more potential uses of the extended friend pattern in metaprogramming contexts, such as using CRTP, see **alexandrescu01**.
- **lakos96**, section 3.6, pp. 136–146, is dedicated to the classic use (and misuse) of friendship.
- A synopsis of the history and process by which the prestandardization form of a `friend` declaration in which the `class` specifier may be omitted is delineated in **miller05**.
- **lakos20** provides extensive advice on *sound* **physical design**, which generally precludes long-distance friendship.

Appendix

Curiously Recurring Template Pattern Use Cases

Refactoring using the curiously recurring template pattern Avoiding code duplication across disparate classes can sometimes be achieved using a strange template pattern first recognized in the mid-90s, which has since become known as the curiously recurring template pattern (CRTP). The pattern is *curious* because it involves the surprising step of declaring as a base class, such as `B`, a template that *expects* the derived class, such as `C`, as a template argument, such as `T`:

```
template <typename T>
class B
{
    // ...
};

class C : public B<C>
{
   // ...
};
```

As a trivial illustration of how the CRTP can be used as a refactoring tool, suppose that we have several classes for which we would like to track, say, just the number of active instances:

```
class A
{
    static int s_count;  // declaration
    // ...

public:
    static int count() { return s_count; }

    A()          { ++s_count; }
    A(const A&)  { ++s_count; }
```