Note, however, that the compiler does not check the commented code, which is easily susceptible to copy-paste and other errors. By uncommenting the code and defaulting it explicitly in class scope, we regain the compiler's syntactic checking of the function signatures without incurring the cost of turning what would have been trivial functions into equivalent non-trivial ones:

```
class C3
{
     // ...

public:
   C3() = default;
       // Create an empty object.

   C3(const C3& rhs) = default;
       // Create an object having the same value as the specified rhs object.

   ~C3() = default;
       // Destroy this object.

   C3& operator=(const C3& rhs) = default;
       // Assign to this object the value of the specified rhs object.
};
```

Preserving type triviality

A particular type being trivial can ~~be beneficial. The~~ type is considered trivial if its default constructor is trivial and it is **trivially copyable** — i.e., it has no non-trivial copy or move constructors, no non-trivial copy or move assignment operators, at least one of those nondeleted, and a trivial destructor. As an example, consider a simple trivial `Metrics` type in the code snippet below containing certain collected metrics for our application:

```
struct Metrics
{
    int d_numRequests;  // number of requests to the service
    int d_numErrors;    // number of error responses

    // All special member functions are generated implicitly.
};
```

Now imagine that we would like to add a constructor to this struct to make its use more convenient: