```
int hoursToSeconds(int hours)
    // ...
{
    return hours * TimeRatios4::k_SECONDS_PER_HOUR;
}

long long hoursToNanos(int hours)
    // Return the number of nanoseconds in the specified hours.  The behavior
    // is undefined unless the result can be represented as a long.
{
    return hours * TimeRatios4::k_NANOS_PER_HOUR;
}
```

In the example above, we've rendered the **constexpr** variables as **static** members of a **struct** rather than placing them at namespace scope primarily to show that, from a user perspective, the two are syntactically indistinguishable, the substantive difference here being that a client would be prevented from unilaterally adding logical content to the "namespace" of a `TimeRatio` **struct**.[5]

Nonintegral symbolic numeric constants

Not all symbolic numeric constants that are needed at compile time are necessarily integral. Consider, for example, the mathematical constants `pi` and `e`, which are typically represented using a floating-point type, such as **double** or **long double**.

The classical solution to avoid encoding this type of constant value as a *magic number* is to instead use a macro, such as is done in the `math.h` header on most operating systems:

```
#define M_E   2.71828182845904523536  /* e */
#define M_PI  3.14159265358979323846 /* pi */

double circumferenceOfCircle(double radius)
{
    return 2 * M_PI * radius;
}
```

While this approach can be effective, it comes with all the well-known downsides of using the C preprocessor, such as potential name collisions. Furthermore, since these constants are not a part of the C or C++ Standard, some platforms do not provide them at all or require additional preprocessor definitions prior to including `math.h` to provide them.

A safer and far less error-prone approach is to instead use a **constexpr** variable for this form of nonintegral constant. Note that, while macros for mathematical constants in `math.h` are defined with sufficiently large precision to be able to initialize variables of possibly

[5] **lakos20**, section 2.4.9, pp. 312–321, specifically Figure 2-23