```
    e_JUL    , e_AUG, e_SEP,
    e_OCT    , e_NOV, e_DEC
};

static_assert(sizeof(Month) == 1 && alignof(Month) == 1, "");
```

With this revised definition of `Month`, the size of a `Date` class is 4 bytes, which is especially valuable for large aggregates:

```
Date timeSeries[1000 * 1000];  // sizeof(timeSeries) is now 4Mb (not 12Mb).
```

Potential Pitfalls

External use of opaque ~~enumerators~~

Providing an explicit underlying type to an **enum** enables clients to declare it as a complete type without its enumerators. Unless the opaque form of its definition is exported in a header file separate from its full definition, external clients wishing to exploit the opaque version will be forced to locally declare it with its **underlying type** but without its enumerator list. If the underlying type of the full definition were to change, any program incorporating *its own* original and now inconsistent elided definition and the *new* full one would become silently **ill formed, no diagnostic required (IFNDR)**. (See Section 2.1."Opaque **enum**s" on page 660.)

Subtleties of integral promotion

When used in an arithmetic context, one might naturally assume that the type of a classic **enum** will first convert to its **underlying type**, which is not always the case. When used in a context that does not explicitly operate on the **enum** type itself, such as a parameter to a function that takes that enum type, **integral promotion** comes into play. For unscoped enumerations without an explicitly specified underlying type and for character types such as **wchar_t**, **char16_t**, and **char32_t**, integral promotion will directly convert the value to the first type in the list **int**, **unsigned int**, **long**, **unsigned long**, **long long**, and **unsigned long long** that is sufficiently large to represent all of the values of the underlying type. Enumerations having a fixed underlying type will, as a first step, behave as if they had decayed to their underlying type.

In most arithmetic expressions, this difference is irrelevant. Subtleties arise, however, when one relies on overload resolution for identifying the underlying type:

```
void f(signed char x);
void f(short x);
void f(int x);
void f(long x);
void f(long long x);
```