If, however, the cardinal value of the `MonthOfYear` enumerators is likely to be relevant to clients, an explicitly scoped *classic* **enum** might be considered as a viable alternative:

```
struct MonthOfYear  // explicit scoping for enum
{
    enum Enum
    {
        e_JAN, e_FEB, e_MAR,  // winter
        e_APR, e_MAY, e_JUN,  // spring
        e_JUL, e_AUG, e_SEP,  // summer
        e_OCT, e_NOV, e_DEC   // autumn
    };
};

bool isSummer(MonthOfYear::Enum month)  // must now pass nested Enum type
{
    return MonthOfYear::e_JUL <= month && month <= MonthOfYear::e_SEP;
}

void doSomethingWithEachMonth()
{
    for (int i =  MonthOfYear::e_JAN;  // iteration variable is now an int
              i <= MonthOfYear::e_DEC;
            ++i)  // OK, convert to underlying type
    {
        // ... (might require cast back to enumerated type)
    }
}
```

Note that such code presumes that the enumerated values will (1) remain in the same order and (2) have contiguous numerical values irrespective of the implementation choice.

External use of opaque ~~enumerators~~

Since scoped enumerations have a UT of **int** by default, clients are always able to (re)declare it, as a **complete type**, without its enumerators. Unless the opaque form of an **enum class**'s definition is exported in a header file separate from the one implementing the publicly accessible full definition, external clients wishing to exploit the opaque version will experience an *attractive nuisance* in that they can provide it locally, along with its underlying type, if any.

If the underlying type of the full definition were to subsequently change, any program incorporating the original elided definition locally and also the new, full one from the header would become silently **ill formed, no diagnostic required (IFNDR)**; see Section 2.1. "Opaque **enum**s" on page 660.