```
void f3()  // Variable templates cannot be defined in functions.
{
    template <typename T> T vt7;         // Error
    template <typename T> static T vt8;  // Error

    vt1<bool> = true;                    // OK
    N::vt3<bool> = true;
    N::vt4<bool> = true;
    S::vt6<bool> = true;
}
```

Like other templates, variable templates may be defined with multiple parameters consisting of arbitrary combinations of type and non-type parameters, including a **parameter pack** in the last position:

```
namespace N
{
    template <typename V, int I, int J> V factor;  // namespace scope
}
```

Variable templates can even be defined recursively, but see *Potential Pitfalls — Recursive variable template initializations require* ***const*** *or* ***constexpr*** on page 163:

```
namespace {
template <int N>
const int sum = N + sum<N - 1>;    // recursive general template

template <> const int sum<0> = 0;  // base case specialization
}  // close unnamed namespace

void f()
{
    std::cout << sum<4> << '\n';  // prints 10
    std::cout << sum<5> << '\n';  // prints 15
    std::cout << sum<6> << '\n';  // prints 21
}
```

Note that while variable templates do not add new functionality, they significantly reduce the boilerplate associated with achieving the same goals without them. For example, compare the definition of `pi` on the previous page with the pre-C++14 code: