```
extern double data[];  // array of unknown size

void f2()
{
    for (double& d : data)  // Error, data is an incomplete type.
    {
        // ...
    }
}

double data[10] = { /*...*/ };  // too late to make the above compile
```

The above example would compile if `data` were declared having a size, e.g., `extern double data[10]`, as that would be a **complete type** and provide sufficient information to traverse the array. The definition of `data` in the example *is* complete but is not visible at the point that the loop is compiled.

An `std::initializer_list` is typically used to initialize an array or container using **braced initialization**; see Section 2.1."Braced Init" on page 215. The `std::initializer_list` template does, however, provide its own `begin` and `end` member functions and is, therefore, directly usable as the range-expression in a range-based `for` loop:

```
#include <initializer_list>  // std::initializer_list

void f3()
{
    for (double v : {1.9, 2.8, 4.7, 7.6, 11.5, 16.4, 22.3, 29.2, 37.1, 46.0})
    {
        // ...
    }
}
```

The example above shows how a series of `double` values can be embedded right within the loop header.

Use Cases

Iterating over all elements of a container

The motivating use case for this feature is looping over the elements in a container:

```
#include <list>  // std::list

void process(int* p);

void f1()
{
    std::list<int> aList{ 1, 2, 4, 7, 11, 16, 22, 29, 37, 46 };
```