```
        public:
            Parser();
            int parse(T* result, const char* input);
                // Load result from null-terminated input; return 0 (on
                // success) or nonzero (with no effect on result).
        };

        template <typename T>
        double analyze(const Parser<T>& parser);
    }
}
```

As suggested by the name `v1`, this namespace serves primarily as a **mechanism** to support library evolution through API and ABI versioning (see *Link-safe ABI versioning* on page 1067 and *Build modes and ABI link safety* on page 1071). The need to specialize **`class`** `Parser` and, independently, the reliance on ADL to find the **free function** template `analyze` require the use of **`inline`** namespaces, as opposed to a conventional namespace followed by a **`using`** directive.

Note that, whenever a subsystem starts out directly in a first-level namespace and is subsequently moved to a second-level nested namespace for the purpose of versioning, declaring the inner namespace **`inline`** is the most **reliable** way to avoid inadvertently destabilizing existing clients; see also *Enabling selective* ***`using`*** *directives for short-named entities* on page 1074.

Now suppose we decide to enhance `parselib` in a non–backwards-compatible manner, such that the **signature** of `parse` takes a second `argument` `size` of type `std::size_t` to allow parsing of non–**null-terminated strings** and to reduce the risk of buffer overruns. Instead of unilaterally removing all support for the previous version in the new release, we can create a second namespace, `v2`, containing the new implementation and then, at some point, make `v2` the **`inline`** namespace instead of `v1`:

```
#include <cstddef>  // std::size_t

namespace parselib
{
    namespace v1  // Notice that v1 is now just a nested namespace.
    {
        template <typename T>
        class Parser
        {
            // ...

        public:
            Parser();
            int parse(T* result, const char* input);
```