Nested Namespaces

C++17 has restructured the way we write nested namespaces, making the syntax more readable.

Namespaces allow grouping types and functions into separate logical units.

For example, it’s common to see that each type or function from a library XY will be stored in a namespace xy. Like in the below case, where there’s `SuperCompressionLib` and it exposes functions called `Compress()` and `Decompress()`:

```
namespace SuperCompressionLib
{
  bool Compress();
  bool Decompress();
}
```

Things get interesting if you have two or more nested namespaces.

```
namespace MySuperCompany {
  namespace SecretProject {
    namespace SafetySystem {
      class SuperArmor {
        // ...
      };
      class SuperShield {
        // ...
      };
    } // SafetySystem
  } // SecretProject
} // MySuperCompany
```

With C++17 nested namespaces can be written in a more compact way:

```
namespace MySuperCompany::SecretProject::SafetySystem {
  class SuperArmor {
    // ...
  };
  class SuperShield {
    // ...
  };
}
```

```
}
```

Such syntax is comfortable, and it will be easier to use for developers that have experience in languages like C# or Java.

In C++17 also the Standard Library was “compacted” in several places by using the new nested namespace feature:

For example, for **regex**.

In C++17 it’s defined as:

```
namespace std::regex_constants {
  typedef T1 syntax_option_type;
  // ...
}
```

Before C++17 the same was declared as:

```
namespace std {
    namespace regex_constants {
        typedef T1 syntax_option_type;
        // ...
    }
}
```

The above nested declarations appear in the C++ Specification, but it might look different in an STL implementation.

> *Extra Info:* The change was proposed in: N4230.

Head over to the next lesson to find out about `__has_include` Preprocessor Expression and compiler support for the concepts studied in this chapter.