```
Tuple<int, double, std::string> tup1(1, 2.0, "three");
    // tup1 holds an int, a double, and an std::string.

Tuple<int, int> tup2(42, 69);
    // tup2 holds two ints.
```

`Tuple` provides a container for a specified set of types, in a manner similar to a **`struct`**, but without the inconvenience of needing to introduce a new **`struct`** definition with its own name. As shown in the example above, the tuples are also initialized correctly according to the specified types (e.g., `tup2` contains two integers initialized, respectively, with `42` and `69`).

In C++03, an approximation to a tuple could be improvised by composing an `std::pair` with itself:

```
#include <utility>  // std::pair
std::pair<int, std::pair<double, long> > v;
    // Define a holder of an int, a double, and a long, accessed as
    // v.first, v.second.first, and v.second.second, respectively.
```

Composite use of `std:pair` types could, in theory, be scaled to arbitrary depth; defining, initializing, and using such types, however, is not always practical. Another approach commonly used in C++03 and similar to the one suggested for the `add` function template above is to define a template class, e.g., `Cpp03Tuple`, having many parameters (e.g, 9), each **defaulted** to a special marker type (e.g., `None`), indicating that the parameter is not used:

```
struct None { };  // empty "tag" used as a special "not used" marker in Cpp03Tuple

template <typename T1 = None, typename T2 = None, typename T3 = None,
          typename T4 = None, typename T5 = None, typename T6 = None,
          typename T7 = None, typename T8 = None, typename T9 = None>
class Cpp03Tuple;
    // struct-like class containing up to 9 data members of arbitrary types
```

`Cpp03Tuple` can be used to store, access, and modify up to nine values together; e.g., `Cpp03Tuple<int, int, std::string>` would consist of two **`int`**s and an `std::string`.

`Cpp03Tuple`'s implementation uses a variety of **metaprogramming** tricks to detect which of the nine type slots are used. This approach is taken by `boost::tuple`,[2] an industrial-strength tuple implemented using C++03-era technology. In contrast, the variadic-template-based declaration (and definition) of a modern C++ tuple is much simpler:

```
template <typename... Ts>
class Cpp11Tuple;  // class template storing an arbitrary sequence of objects
```

C++11 introduced the Standard Library class template `std::tuple`, declared in a manner similar to `Cpp11Tuple`.

[2] https://github.com/boostorg/tuple/blob/develop/include/boost/tuple/tuple.hpp