## List Initialization: `std::initializer_list<T>`

The C++ Standard Library's `std::initializer_list` class template supports lightweight, compiler-generated arrays of nonmodifiable values that are initialized in source code similarly to built-in, C-style arrays using the generalized braced initialization syntax.

### Description

C++, ~~and~~ C before it, allows built-in arrays to be initialized via brace-enclosed lists of values:

```
int data[] = { 0, 1, 1, 2, 3, 5, 8, 13 };  // initializer list of 8 int values
```

C++11 extends this concept to allow such lists of values to be provided to **user-defined types (UDTs)** in a variety of circumstances. The compiler arranges for the values to be stored in an unnamed C-style array of **`const`** elements and provides access to that array via an object of type `std::initializer_list` instantiated on the element type. This object is a *lightweight* proxy to the elements of the array that provides a familiar API to both iterate over the elements of the array and query its size. Note that copying the `std::initializer_list` object does not copy the array elements. The C++ Standard provides a reference definition that comprises **`typedef`**s, accessors, and an explicitly declared default constructor, along with implicit definitions of the other five **special member functions**; see Section 1.1."Defaulted Functions" on page 33:

```
namespace std
{

template <typename E>
class initializer_list  // illustration of programmer-accessible interface
{

public:
    typedef E value_type;               // C++ type of each array element
    typedef const E& reference;         // There is no nonconst reference.
    typedef const E& const_reference;   // const lvalue reference type
    typedef size_t size_type;           // type returned by size()

    typedef const E* iterator;          // There is no nonconst iterator.
    typedef const E* const_iterator;    // const element-iterator type

    constexpr initializer_list() noexcept;  // default constructor

    constexpr size_t size() const noexcept;     // number of elements
    constexpr const E* begin() const noexcept;  // beginning iterator
    constexpr const E* end() const noexcept;    // one-past-the-last iterator
};
```