```
    assert(x != il);
    assert( x.isEqual({ 1, 2, 3, 4, 5, 6 }));

    x.shrink(2);

    std::initializer_list<int> ilB = { 1, 2 };

    assert(!x.isEqual({ 1, 2, 3, 4, 5, 6 }));
    assert( x.isEqual({ 1, 2 }));
    assert( x.isEqual(ilB));
    assert(x == ilB);

    assert((x == { 1, 2 }));  // Error, not an std::initializer_list

    x += { 8, 9, 10 };

    assert(!x.isEqual({ 1, 2 }));
    assert( x.isEqual({ 1, 2, 8, 9, 10 }));
}
```

Note that due to a quirk of the C++ grammar for certain operators, ~~using *braced-init-list* as the right-hand side of the argument of an **operator**== does not implicitly deduce an *initializer-list*, even if the **operator**== takes an *initializer-list* as the parameter~~; see Section 2.1."Braced Init" on page 215.

Functions consuming a variable number of arguments of the same type

Suppose we want a function that takes an arbitrary number of arguments all of the same type. In our example, we write a function that concatenates a number of input strings together separated by commas:

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

std::string concatenate(std::initializer_list<std::string> ils)
{
    std::string separator;
    std::string result;
    for (const std::string* p = ils.begin(); p != ils.end(); ++p)
    {
        result.append(separator);
        result.append(*p);
        separator = ",";
    }
    return result;
}

std::string hex_digits = concatenate({"A", "B", "C", "D", "E"});
```