$Id: xless.mm,v 1.8 2022-02-06 18:32:08-08 - - $
/afs/cats.ucsc.edu/courses/cse111-wm/Assignments/asg3-listmap-templates/dissection-misc.d
https://www2.ucsc.edu/courses/cse111-wm/:/Assignments/asg3-listmap-templates/dissection-misc.d/

## 1. struct xless, struct xgreater

In this example we show how to use a single comparator to perform all comparisons. Consider the function object **`xless`**.

```
template <typename Type>
struct xless {
   bool operator() (const Type &left, const Type &right) const {
      return left < right;
   }
};
```

When instantiated as an object, it behaves as would any function. **`operator()`** in C++ is special in that it can be written to take any number of arguments. Other operators in C++ are restricted to the number of arguments given by the syntax.

**`xless<string>`** can be passed into any function as a comparator strings. Similarly, **`xless<int>`** can be passed in as a function object for integers. It is not possible syntactically to use an operator as an argument.

Example: **`f(operator<)`** is just a syntax error. But **`f(xless())`** can be used for the same purpose.

Also consider sorting in the reverse order.

```
template <typename Type>
struct xgreater {
   bool operator() (const Type &left, const Type &right) const {
      return left > right;
   }
};
```

## 2. Sorting

**`sort`** is one of the algorithms in the standard library. If we have

```
vector<string> vs {"hello", "world", "foo", "bar", "baz"};
vector<int> vi {3, 1, 4, 55, 77, -8};
```

They could be sorted as follows:

```
sort (vs.begin(), vs.end());
sort (vs.begin(), vs.end(), xgreater<string>());
sort (vi.begin(), vi.end(), xgreater<int>());
sort (vi.begin(), vi.end(), xless<int>());
```

In the first case **`sort`** will sort in increasing order by default. In the other cases, a function object is passed into sort to determine ordering. In fact any comparison might be used, such as alphabetical rather than lexicographcc order.

If names are to be sorted, a slightly more complicated sorter might be used, such as one that compares last names, and only considers first names when last names are the same.

See the examples **`test-xless.cpp`** and **`sorting.cpp`** in the **`misc/`** subdirectory.

If we have, for example

```
struct name {
   string last;
   string first;
};
bool operator< (const name& one, const name& two) {
   if (one.last < two.last) return true;
   if (one.last > two.last) return false;
   return one.first < two.first;
}
```

then **`xless<name>()`** could be passed into the sorting function, as in

```
sort (vn.begin(), vn.end(), xless<name>());
```