```
 1: // $Id: reverse.cpp,v 1.31 2015-02-05 19:10:01-08 - - $
 2:
 3: //
 4: // Show how to reverse a collection using a pair of iterators.
 5: // Reverse requires a bidirectionaly iterator, with ++ and --.
 6: // Also show that pointers can be used as iterators.
 7: //
 8:
 9: #include <iostream>
10: #include <list>
11: #include <typeinfo>
12: #include <vector>
13:
14: #include <cxxabi.h>
15:
16: using namespace std;
17:
18: template <typename itor_t>
19: void xswap (const itor_t &itor1, const itor_t &itor2) {
20:    typeof (*itor1) tmp = std::move (*itor1);
21:    *itor1 = std::move (*itor2);
22:    *itor2 = std::move (tmp);
23: }
24:
25: // Reverse a collection from each end.
26: // Uses a bidirectional iterator.
27: // Xreverse avoids clash with <algorithm>reverse
28: // Pass itors by value so we don't need an extra local copy.
29: template <typename biitor_t>
30: void xreverse (biitor_t begin, biitor_t end) {
31:    while (begin != end && begin != --end) {
32:       xswap (begin, end);
33:       ++begin;
34:    }
35: }
36:
```

```
37:
38: // Print a range using an iterator.
39: template <typename itor_t>
40: void print (const itor_t &begin, const itor_t &end) {
41:    cout << "  ";
42:    for (itor_t itor = begin; itor != end; ++itor) cout << " " << *itor;
43:    cout << endl;
44: }
45:
46: // Demangle typeinfo names
47: void print_type (const char *const label, const type_info &info) {
48:    const char *const name = info.name();
49:    cout << "   typeid(" << label << ") = " << name << endl;
50:    int status;
51:    char *unmangled = abi::__cxa_demangle (name, 0, 0, &status);
52:    cout << "      unmangled = " << unmangled << endl;
53:    free (unmangled); // C code allocated by malloc.
54: }
55:
56: // Print, reverse, print, reverse.
57: template <typename itor_t>
58: void print_reverse_twice (const itor_t &begin, const itor_t &end) {
59:    print_type ("itor", typeid (typeof (end)));
60:    print_type ("*itor", typeid (typeof (*end)));
61:    for (int count = 0; count < 2; ++count) {
62:       print (begin, end);
63:       xreverse (begin, end);
64:    }
65: }
66:
67: // Load container with data.
68: // itor_t must be convertable to container_t::const_iterator.
69: template <typename container_t, typename itor_t>
70: void load_container (container_t &cont, itor_t begin, itor_t end) {
71:    for (itor_t itor = begin; itor != end; ++itor) {
72:       cont.push_back (*itor);
73:    }
74: }
75:
```

```
 76:
 77: // Put argv into a vector, xreverse, then print.
 78: // Use iterator-style to process argv.
 79: void testvector (char **argv, char **argend) {
 80:    cout << endl << __func__ << ":" << endl;
 81:    vector<string> vec;
 82:    load_container (vec, argv, argend);
 83:    print_reverse_twice (vec.begin(), vec.end());
 84: }
 85:
 86: // Put argv into a vector, xreverse, then print.
 87: // Use iterator-style to process argv.
 88: // NOTE: Same code, just a different data structure.
 89: void testlist (char **argv, char **argend) {
 90:    cout << endl << __func__ << ":" << endl;
 91:    list<string> lis;
 92:    load_container (lis, argv, argend);
 93:    print_reverse_twice (lis.begin(), lis.end());
 94: }
 95:
 96: // Now actually just use real pointers as iterators.
 97: // NOTE: Same code, just an array instead of iterators.
 98: void testcharstar (char **argv, char **argend) {
 99:    cout << endl << __func__ << ":" << endl;
100:    print_reverse_twice (argv, argend);
101: }
102:
103: // Now actually just use an array of ints.
104: // NOTE: Same code, just an array instead of iterators.
105: void testintarray () {
106:    cout << endl << __func__ << ":" << endl;
107:    int array[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
108:    int *end = array + sizeof array / sizeof *array;
109:    print_reverse_twice (&*array, end);
110: }
111:
112: // Main program to call each one in turn.
113: int main (int argc, char **argv) {
114:    testvector (argv + 1, argv + argc);
115:    testlist (argv + 1, argv + argc);
116:    testcharstar (argv + 1, argv + argc);
117:    testintarray ();
118:    return 0;
119: }
120:
121: //TEST// ./reverse foo bar baz qux hello world >reverse.out 2>&1
122: //TEST// mkpspdf reverse.ps reverse.cpp* reverse.out
123:
```

```
    1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting reverse.cpp
    2: reverse.cpp:
    3:       $Id: reverse.cpp,v 1.31 2015-02-05 19:10:01-08 - - $
    4: g++ -g -O0 -Wall -Wextra -rdynamic -std=gnu++11 reverse.cpp -o reverse -
lglut -lGLU -lGL -lX11 -lrt -lm
    5: rm -f reverse.o
    6: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished reverse.cpp
```

```
    1:
    2: testvector:
    3:    typeid(itor) = N9__gnu_cxx17__normal_iteratorIPSsSt6vectorISsSaISsEEE
E
    4:       unmangled = __gnu_cxx::__normal_iterator<std::string*, std::vector
<std::string, std::allocator<std::string> > >
    5:    typeid(*itor) = Ss
    6:       unmangled = std::string
    7:    foo bar baz qux hello world
    8:    world hello qux baz bar foo
    9:
   10: testlist:
   11:    typeid(itor) = St14_List_iteratorISsE
   12:       unmangled = std::_List_iterator<std::string>
   13:    typeid(*itor) = Ss
   14:       unmangled = std::string
   15:    foo bar baz qux hello world
   16:    world hello qux baz bar foo
   17:
   18: testcharstar:
   19:    typeid(itor) = PPc
   20:       unmangled = char**
   21:    typeid(*itor) = Pc
   22:       unmangled = char*
   23:    foo bar baz qux hello world
   24:    world hello qux baz bar foo
   25:
   26: testintarray:
   27:    typeid(itor) = Pi
   28:       unmangled = int*
   29:    typeid(*itor) = i
   30:       unmangled = int
   31:    3 1 4 1 5 9 2 6 5 3 5
   32:    5 3 5 6 2 9 5 1 4 1 3
```