```
 1: // $Id: treefree.cpp,v 1.71 2015-03-26 22:15:48-07 - - $
 2:
 3: // Shared_ptrs use reference counting in order to automatically
 4: // free objects, but that does not work for cyclic data structures.
 5: // This illustrates how to avoid the problem.
 6:
 7: #include <iomanip>
 8: #include <iostream>
 9: #include <map>
10: #include <memory>
11: using namespace std;
12:
13: //////////////////////////////////////////////////////////////////
14: // tree.h
15: //////////////////////////////////////////////////////////////////
16:
17: class tree;
18: using tree_ptr = shared_ptr<tree>;
19: using tree_dir = map<string,tree_ptr>;
20: using tree_itor = tree_dir::iterator;
21:
22: class tree {
23:       friend ostream& operator<< (ostream&, const tree*);
24:    private:
25:       static size_t next_seq;
26:       size_t seq;
27:       tree_dir data;
28:       void print (size_t);
29:       void disown (size_t);
30:    public:
31:       static const string PARENT;
32:       static tree_ptr make_root();
33:       static tree_ptr make (tree_ptr ptr);
34:       tree (tree_ptr parent);
35:       ~tree();
36:       void emplace (const tree_dir::key_type&,
37:                     const tree_dir::mapped_type&);
38:       const tree_itor begin() { return data.begin(); }
39:       const tree_itor end() { return data.end(); }
40:       void print() { print (0); }
41:       void disown() { disown (0); }
42: };
43:
```

```
44:
45: //////////////////////////////////////////////////////////////////
46: // tree.cpp
47: //////////////////////////////////////////////////////////////////
48:
49: size_t tree::next_seq {0};
50: const string tree::PARENT = "..";
51:
52: ostream& operator<< (ostream& out, const tree* ptr) {
53:    if (ptr == nullptr) return out << "nullptr";
54:                   else return out << "[" << ptr->seq << "]"
55:                                   << static_cast<const void*> (ptr);
56: }
57:
58: tree::tree (tree_ptr parent): seq(next_seq++), data({{PARENT,parent}}) {
59:    cout << this << "->" << __func__ << "(" << parent << ")" << endl;
60: }
61:
62: tree::~tree() {
63:    cout << this << "->" << __func__ << "()" << endl;
64: }
65:
66: void tree::emplace (const tree_dir::key_type& key,
67:                     const tree_dir::mapped_type& value) {
68:    data.emplace (key, value);
69: }
70:
71: void tree::disown (size_t depth) {
72:    cout << __func__ << ": " << setw (depth * 3) << "" << this << endl;
73:    data.erase (PARENT);
74:    for (auto n: data) n.second->disown (depth + 1);
75: }
76:
77: // Depth-first pre-order traversal.
78: void tree::print (size_t depth) {
79:    for (const auto itor: data) {
80:       cout << __func__ << ": " << setw (depth * 3) << "" << this
81:            << ": \"" << itor.first << "\" -> " << itor.second
82:            << " (" << itor.second.use_count() << ")" << endl;
83:       if (itor.first != PARENT and itor.second != nullptr) {
84:          itor.second->print (depth + 1);
85:       }
86:    }
87: }
88:
89: tree_ptr tree::make_root() {
90:    tree_ptr ptr = make_shared<tree> (nullptr);
91:    ptr->data[PARENT] = ptr;
92:    return ptr;
93: }
94:
95: tree_ptr tree::make (tree_ptr parent) {
96:    if (parent == nullptr) throw logic_error ("tree::make(nullptr)");
97:    return make_shared<tree> (parent);
98: }
99:
```

```
100:
101: //////////////////////////////////////////////////////////////////
102: // main.cpp
103: //////////////////////////////////////////////////////////////////
104:
105: int main (int argc, char** argv) {
106:    (void) argc;
107:    (void) argv;
108:    shared_ptr<tree> root = tree::make_root();
109:    root->emplace ("foo", tree::make (root));
110:    root->emplace ("bar", tree::make (root));
111:    for (auto itor: *root) {
112:       if (itor.first == tree::PARENT) continue;
113:       for (int count = 0; count < 3; ++count) {
114:          string quux = "qux";
115:          quux.insert (1, count, 'u');
116:          itor.second->emplace (quux, tree::make (itor.second));
117:       }
118:    }
119:    cout << "[seq]address: key -> value (use count)" << endl;
120:    root->print();
121:    root->disown();
122:    return 0;
123: }
124:
125: //TEST// alias grind='valgrind --leak-check=full --show-reachable=yes'
126: //TEST// grind treefree >treefree.out 2>treefree.ground
127: //TEST// mkpspdf treefree.ps treefree.cpp* treefree.out treefree.ground
128:
```

```
   1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting treefree.cpp
   2: treefree.cpp:
   3:       $Id: treefree.cpp,v 1.71 2015-03-26 22:15:48-07 - - $
   4: g++ -g -O0 -Wall -Wextra -rdynamic -std=gnu++11 treefree.cpp -o treefree
-lglut -lGLU -lGL -lX11 -lrt -lm
   5: rm -f treefree.o
   6: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished treefree.cpp
```

```
 1: [0]0x4e7d108->tree(nullptr)
 2: [1]0x4e7d218->tree([0]0x4e7d108)
 3: [2]0x4e7d408->tree([0]0x4e7d108)
 4: [3]0x4e7d658->tree([2]0x4e7d408)
 5: [4]0x4e7d8a8->tree([2]0x4e7d408)
 6: [5]0x4e7daf8->tree([2]0x4e7d408)
 7: [6]0x4e7dce8->tree([1]0x4e7d218)
 8: [7]0x4e7df38->tree([1]0x4e7d218)
 9: [8]0x4e7e188->tree([1]0x4e7d218)
10: [seq]address: key -> value (use count)
11: print: [0]0x4e7d108: ".." -> [0]0x4e7d108 (5)
12: print: [0]0x4e7d108: "bar" -> [2]0x4e7d408 (5)
13: print:    [2]0x4e7d408: ".." -> [0]0x4e7d108 (5)
14: print:    [2]0x4e7d408: "quuux" -> [5]0x4e7daf8 (2)
15: print:       [5]0x4e7daf8: ".." -> [2]0x4e7d408 (6)
16: print:    [2]0x4e7d408: "quux" -> [4]0x4e7d8a8 (2)
17: print:       [4]0x4e7d8a8: ".." -> [2]0x4e7d408 (6)
18: print:    [2]0x4e7d408: "qux" -> [3]0x4e7d658 (2)
19: print:       [3]0x4e7d658: ".." -> [2]0x4e7d408 (6)
20: print: [0]0x4e7d108: "foo" -> [1]0x4e7d218 (5)
21: print:    [1]0x4e7d218: ".." -> [0]0x4e7d108 (5)
22: print:    [1]0x4e7d218: "quuux" -> [8]0x4e7e188 (2)
23: print:       [8]0x4e7e188: ".." -> [1]0x4e7d218 (6)
24: print:    [1]0x4e7d218: "quux" -> [7]0x4e7df38 (2)
25: print:       [7]0x4e7df38: ".." -> [1]0x4e7d218 (6)
26: print:    [1]0x4e7d218: "qux" -> [6]0x4e7dce8 (2)
27: print:       [6]0x4e7dce8: ".." -> [1]0x4e7d218 (6)
28: disown: [0]0x4e7d108
29: disown:    [2]0x4e7d408
30: disown:       [5]0x4e7daf8
31: disown:       [4]0x4e7d8a8
32: disown:       [3]0x4e7d658
33: disown:    [1]0x4e7d218
34: disown:       [8]0x4e7e188
35: disown:       [7]0x4e7df38
36: disown:       [6]0x4e7dce8
37: [0]0x4e7d108->~tree()
38: [1]0x4e7d218->~tree()
39: [6]0x4e7dce8->~tree()
40: [7]0x4e7df38->~tree()
41: [8]0x4e7e188->~tree()
42: [2]0x4e7d408->~tree()
43: [3]0x4e7d658->~tree()
44: [4]0x4e7d8a8->~tree()
45: [5]0x4e7daf8->~tree()
```

```
    1: ==14412== Memcheck, a memory error detector
    2: ==14412== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al
.
    3: ==14412== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright i
nfo
    4: ==14412== Command: treefree
    5: ==14412==
    6: ==14412==
    7: ==14412== HEAP SUMMARY:
    8: ==14412==     in use at exit: 0 bytes in 0 blocks
    9: ==14412==   total heap usage: 40 allocs, 40 frees, 2,056 bytes allocated
   10: ==14412==
   11: ==14412== All heap blocks were freed -- no leaks are possible
   12: ==14412==
   13: ==14412== For counts of detected and suppressed errors, rerun with: -v
   14: ==14412== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```