

```
1: // $Id: treefree.cpp,v 1.75 2019-04-22 15:56:04-07 - - $
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:     explicit 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: // tree.cpp
46: //////////////////////////////////////
47:
48: size_t tree::next_seq {0};
49: const string tree::PARENT = "..";
50:
```

```
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 << "->" << __PRETTY_FUNCTION__
60:         << "(" << parent << ")" << endl;
61: }
62:
63: tree::~~tree() {
64:     cout << this << "->" << __PRETTY_FUNCTION__ << "()" << endl;
65: }
66:
67: void tree::emplace (const tree_dir::key_type& key,
68:                     const tree_dir::mapped_type& value) {
69:     data.emplace (key, value);
70: }
71:
72: void tree::disown (size_t depth) {
73:     cout << __PRETTY_FUNCTION__ << ": "
74:         << setw (depth * 3) << "" << this << endl;
75:     data.erase (PARENT);
76:     for (auto n: data) n.second->disown (depth + 1);
77: }
78:
79: // Depth-first pre-order traversal.
80: void tree::print (size_t depth) {
81:     for (const auto itor: data) {
82:         cout << __PRETTY_FUNCTION__ << ": "
83:             << setw (depth * 3) << "" << this
84:             << ": \"" << itor.first << "\" -> " << itor.second
85:             << " (" << itor.second.use_count() << ")" << endl;
86:         if (itor.first != PARENT and itor.second != nullptr) {
87:             itor.second->print (depth + 1);
88:         }
89:     }
90: }
91:
92: tree_ptr tree::make_root() {
93:     tree_ptr ptr = make_shared<tree> (nullptr);
94:     ptr->data[PARENT] = ptr;
95:     return ptr;
96: }
97:
98: tree_ptr tree::make (tree_ptr parent) {
99:     if (parent == nullptr) throw logic_error ("tree::make(nullptr)");
100:    return make_shared<tree> (parent);
101: }
102:
```

```
103:
104: //////////////////////////////////////
105: // main.cpp
106: //////////////////////////////////////
107:
108: int main (int argc, char** argv) {
109:     (void) argc;
110:     (void) argv;
111:     shared_ptr<tree> root = tree::make_root();
112:     root->emplace ("foo", tree::make (root));
113:     root->emplace ("bar", tree::make (root));
114:     for (auto itor: *root) {
115:         if (itor.first == tree::PARENT) continue;
116:         for (int count = 0; count < 3; ++count) {
117:             string quux = "qux";
118:             quux.insert (1, count, 'u');
119:             itor.second->emplace (quux, tree::make (itor.second));
120:         }
121:     }
122:     cout << "[seq]address: key -> value (use count)" << endl;
123:     root->print();
124:     root->disown();
125:     return 0;
126: }
127:
128: //TEST// alias grind='valgrind --leak-check=full --show-reachable=yes'
129: //TEST// grind treefree >treefree.out 2>treefree.ground
130: //TEST// mkpspdf treefree.ps treefree.cpp* treefree.out treefree.ground
131:
```

[illegible]

```
1: [0]0x5a240b0->tree::tree(tree_ptr) (nullptr)
2: [1]0x5a241c0->tree::tree(tree_ptr) ([0]0x5a240b0)
3: [2]0x5a243b0->tree::tree(tree_ptr) ([0]0x5a240b0)
4: [3]0x5a24600->tree::tree(tree_ptr) ([2]0x5a243b0)
5: [4]0x5a24850->tree::tree(tree_ptr) ([2]0x5a243b0)
6: [5]0x5a24aa0->tree::tree(tree_ptr) ([2]0x5a243b0)
7: [6]0x5a24c90->tree::tree(tree_ptr) ([1]0x5a241c0)
8: [7]0x5a24ee0->tree::tree(tree_ptr) ([1]0x5a241c0)
9: [8]0x5a25130->tree::tree(tree_ptr) ([1]0x5a241c0)
10: [seq]address: key -> value (use count)
11: void tree::print(size_t): [0]0x5a240b0: "." -> [0]0x5a240b0 (5)
12: void tree::print(size_t): [0]0x5a240b0: "bar" -> [2]0x5a243b0 (5)
13: void tree::print(size_t): [2]0x5a243b0: "." -> [0]0x5a240b0 (5)
14: void tree::print(size_t): [2]0x5a243b0: "quux" -> [5]0x5a24aa0 (2)
15: void tree::print(size_t): [5]0x5a24aa0: "." -> [2]0x5a243b0 (6)
16: void tree::print(size_t): [2]0x5a243b0: "quux" -> [4]0x5a24850 (2)
17: void tree::print(size_t): [4]0x5a24850: "." -> [2]0x5a243b0 (6)
18: void tree::print(size_t): [2]0x5a243b0: "qux" -> [3]0x5a24600 (2)
19: void tree::print(size_t): [3]0x5a24600: "." -> [2]0x5a243b0 (6)
20: void tree::print(size_t): [0]0x5a240b0: "foo" -> [1]0x5a241c0 (5)
21: void tree::print(size_t): [1]0x5a241c0: "." -> [0]0x5a240b0 (5)
22: void tree::print(size_t): [1]0x5a241c0: "quux" -> [8]0x5a25130 (2)
23: void tree::print(size_t): [8]0x5a25130: "." -> [1]0x5a241c0 (6)
24: void tree::print(size_t): [1]0x5a241c0: "quux" -> [7]0x5a24ee0 (2)
25: void tree::print(size_t): [7]0x5a24ee0: "." -> [1]0x5a241c0 (6)
26: void tree::print(size_t): [1]0x5a241c0: "qux" -> [6]0x5a24c90 (2)
27: void tree::print(size_t): [6]0x5a24c90: "." -> [1]0x5a241c0 (6)
28: void tree::disown(size_t): [0]0x5a240b0
29: void tree::disown(size_t): [2]0x5a243b0
30: void tree::disown(size_t): [5]0x5a24aa0
31: void tree::disown(size_t): [4]0x5a24850
32: void tree::disown(size_t): [3]0x5a24600
33: void tree::disown(size_t): [1]0x5a241c0
34: void tree::disown(size_t): [8]0x5a25130
35: void tree::disown(size_t): [7]0x5a24ee0
36: void tree::disown(size_t): [6]0x5a24c90
37: [0]0x5a240b0->tree::~~tree() ()
38: [1]0x5a241c0->tree::~~tree() ()
39: [6]0x5a24c90->tree::~~tree() ()
40: [7]0x5a24ee0->tree::~~tree() ()
41: [8]0x5a25130->tree::~~tree() ()
42: [2]0x5a243b0->tree::~~tree() ()
43: [3]0x5a24600->tree::~~tree() ()
44: [4]0x5a24850->tree::~~tree() ()
45: [5]0x5a24aa0->tree::~~tree() ()
```

```
1: ==22345== Memcheck, a memory error detector
2: ==22345== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
3: ==22345== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
4: ==22345== Command: treefree
5: ==22345==
6: ==22345==
7: ==22345== HEAP SUMMARY:
8: ==22345==      in use at exit: 0 bytes in 0 blocks
9: ==22345==    total heap usage: 39 allocs, 39 frees, 1,975 bytes allocated
10: ==22345==
11: ==22345== All heap blocks were freed -- no leaks are possible
12: ==22345==
13: ==22345== For counts of detected and suppressed errors, rerun with: -v
14: ==22345== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```