```
 1: // $Id: treefree.cpp,v 1.89 2020-11-04 19:27:34-08 - - $
 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 <algorithm>
 8: #include <iomanip>
 9: #include <iostream>
10: #include <map>
11: #include <memory>
12: using namespace std;
13:
14: //////////////////////////////////////////////////////////////////
15: // tree.h
16: //////////////////////////////////////////////////////////////////
17:
18: class tree;
19: using tree_ptr = shared_ptr<tree>;
20: using tree_dir = map<string,tree_ptr>;
21: using tree_itor = tree_dir::iterator;
22:
23: class tree {
24:       friend ostream& operator<< (ostream&, const tree*);
25:    private:
26:       static size_t next_seq;
27:       size_t seq;
28:       tree_dir data;
29:       void print (size_t);
30:       void disown (size_t);
31:    public:
32:       static const string PARENT;
33:       static tree_ptr make_root();
34:       static tree_ptr make (tree_ptr ptr);
35:       explicit tree (tree_ptr parent);
36:       ~tree();
37:       void emplace (const tree_dir::key_type&,
38:                     const tree_dir::mapped_type&);
39:       const tree_itor begin() { return data.begin(); }
40:       const tree_itor end() { return data.end(); }
41:       void print() { print (0); }
42:       void disown() { disown (0); }
43: };
44:
45: //////////////////////////////////////////////////////////////////
46: // tree.cpp
47: //////////////////////////////////////////////////////////////////
48:
49: size_t tree::next_seq {0};
50: const string tree::PARENT = "..";
51:
```

```
 52:
 53: ostream& operator<< (ostream& out, const tree* ptr) {
 54:    if (ptr == nullptr) return out << "nullptr";
 55:                   else return out << "[" << ptr->seq << "]"
 56:                               << static_cast<const void*> (ptr);
 57: }
 58:
 59: tree::tree (tree_ptr parent): seq(next_seq++), data({{PARENT,parent}}) {
 60:    cout << this << "->" << __PRETTY_FUNCTION__
 61:         << ": parent=" << parent << endl;
 62: }
 63:
 64: tree::~tree() {
 65:    cout << this << "->" << __PRETTY_FUNCTION__ << ":";
 66:    for (const auto& pair: data) cout << " " << pair.first;
 67:    cout << endl;
 68: }
 69:
 70: void tree::emplace (const tree_dir::key_type& key,
 71:                     const tree_dir::mapped_type& value) {
 72:    data.emplace (key, value);
 73: }
 74:
 75: void tree::disown (size_t depth) {
 76:    cout << __PRETTY_FUNCTION__ << ": "
 77:         << setw (depth * 3) << "" << this << endl;
 78:    data.erase (PARENT);
 79:    for (auto pair: data) pair.second->disown (depth + 1);
 80: }
 81:
 82: // Depth-first pre-order traversal.
 83: void tree::print (size_t depth) {
 84:    for (const auto itor: data) {
 85:       cout << __PRETTY_FUNCTION__ << ": "
 86:            << setw (depth * 3) << "" << this
 87:            << ": \"" << itor.first << "\" -> " << itor.second
 88:            << " (" << itor.second.use_count() << ")" << endl;
 89:       if (itor.first != PARENT and itor.second != nullptr) {
 90:          itor.second->print (depth + 1);
 91:       }
 92:    }
 93: }
 94:
 95: tree_ptr tree::make_root() {
 96:    tree_ptr ptr = make_shared<tree> (nullptr);
 97:    ptr->data[PARENT] = ptr;
 98:    return ptr;
 99: }
100:
101: tree_ptr tree::make (tree_ptr parent) {
102:    if (parent == nullptr) throw logic_error ("tree::make(nullptr)");
103:    return make_shared<tree> (parent);
104: }
105:
```

```
106:
107: //////////////////////////////////////////////////////////////////
108: // main.cpp
109: //////////////////////////////////////////////////////////////////
110:
111: int main (int argc, char** argv) {
112:    cout << "Command:";
113:    for_each (&argv[0], &argv[argc], [](char* arg){cout << " " << arg;});
114:    cout << endl;
115:
116:    bool want_disown = argc > 1 and argv[1] == string ("-d");
117:    shared_ptr<tree> root = tree::make_root();
118:    root->emplace ("foo", tree::make (root));
119:    root->emplace ("bar", tree::make (root));
120:    for (auto itor: *root) {
121:       if (itor.first == tree::PARENT) continue;
122:       for (int count = 0; count < 3; ++count) {
123:          string quux = "qux";
124:          quux.insert (1, count, 'u');
125:          itor.second->emplace (quux, tree::make (itor.second));
126:       }
127:    }
128:    cout << "[seq]address: key -> value (use count)" << endl;
129:    root->print();
130:    if (want_disown) root->disown();
131:    return 0;
132: }
133:
134: //TEST// valgrind treefree -0 >treefree.out-0 2>&1
135: //TEST// valgrind treefree -d >treefree.out-d 2>&1
136: //TEST// mkpspdf treefree.ps treefree.cpp* treefree.out*
137:
```

```
    1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting treefree.cpp
    2: $Id: treefree.cpp,v 1.89 2020-11-04 19:27:34-08 - - $
    3: checksource treefree.cpp
    4: checksource: TOTAL COMPLAINTS: 0
    5: treefree.cpp: $Id: treefree.cpp,v 1.89 2020-11-04 19:27:34-08 - - $
    6: cpplint.py.perl treefree.cpp
    7: Done processing treefree.cpp
    8: g++ -Wall -Wextra -Wpedantic -Wshadow -fdiagnostics-color=never -std=gnu
++2a -Wold-style-cast -g -O0 treefree.cpp -o treefree -lm
    9: rm -f treefree.o
   10: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished treefree.cpp
```

```
    1: ==32448== Memcheck, a memory error detector
    2: ==32448== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
    3: ==32448== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
    4: ==32448== Command: treefree -0
    5: ==32448==
    6: Command: treefree -0
    7: [0]0x5a24110->tree::tree(tree_ptr): parent=nullptr
    8: [1]0x5a24220->tree::tree(tree_ptr): parent=[0]0x5a24110
    9: [2]0x5a24410->tree::tree(tree_ptr): parent=[0]0x5a24110
   10: [3]0x5a24660->tree::tree(tree_ptr): parent=[2]0x5a24410
   11: [4]0x5a248b0->tree::tree(tree_ptr): parent=[2]0x5a24410
   12: [5]0x5a24b00->tree::tree(tree_ptr): parent=[2]0x5a24410
   13: [6]0x5a24cf0->tree::tree(tree_ptr): parent=[1]0x5a24220
   14: [7]0x5a24f40->tree::tree(tree_ptr): parent=[1]0x5a24220
   15: [8]0x5a25190->tree::tree(tree_ptr): parent=[1]0x5a24220
   16: [seq]address: key -> value (use count)
   17: void tree::print(size_t): [0]0x5a24110: ".." -> [0]0x5a24110 (5)
   18: void tree::print(size_t): [0]0x5a24110: "bar" -> [2]0x5a24410 (5)
   19: void tree::print(size_t):    [2]0x5a24410: ".." -> [0]0x5a24110 (5)
   20: void tree::print(size_t):    [2]0x5a24410: "quuux" -> [5]0x5a24b00 (2)
   21: void tree::print(size_t):       [5]0x5a24b00: ".." -> [2]0x5a24410 (6)
   22: void tree::print(size_t):    [2]0x5a24410: "quux" -> [4]0x5a248b0 (2)
   23: void tree::print(size_t):       [4]0x5a248b0: ".." -> [2]0x5a24410 (6)
   24: void tree::print(size_t):    [2]0x5a24410: "qux" -> [3]0x5a24660 (2)
   25: void tree::print(size_t):       [3]0x5a24660: ".." -> [2]0x5a24410 (6)
   26: void tree::print(size_t): [0]0x5a24110: "foo" -> [1]0x5a24220 (5)
   27: void tree::print(size_t):    [1]0x5a24220: ".." -> [0]0x5a24110 (5)
   28: void tree::print(size_t):    [1]0x5a24220: "quuux" -> [8]0x5a25190 (2)
   29: void tree::print(size_t):       [8]0x5a25190: ".." -> [1]0x5a24220 (6)
   30: void tree::print(size_t):    [1]0x5a24220: "quux" -> [7]0x5a24f40 (2)
   31: void tree::print(size_t):       [7]0x5a24f40: ".." -> [1]0x5a24220 (6)
   32: void tree::print(size_t):    [1]0x5a24220: "qux" -> [6]0x5a24cf0 (2)
   33: void tree::print(size_t):       [6]0x5a24cf0: ".." -> [1]0x5a24220 (6)
   34: ==32448==
   35: ==32448== HEAP SUMMARY:
   36: ==32448==     in use at exit: 1,863 bytes in 35 blocks
   37: ==32448==   total heap usage: 40 allocs, 5 frees, 2,002 bytes allocated
   38: ==32448==
   39: ==32448== LEAK SUMMARY:
   40: ==32448==    definitely lost: 72 bytes in 1 blocks
   41: ==32448==    indirectly lost: 1,764 bytes in 33 blocks
   42: ==32448==      possibly lost: 0 bytes in 0 blocks
   43: ==32448==    still reachable: 27 bytes in 1 blocks
   44: ==32448==                       of which reachable via heuristic:
   45: ==32448==                         stdstring          : 27 bytes in 1 blo
cks
   46: ==32448==         suppressed: 0 bytes in 0 blocks
   47: ==32448== Rerun with --leak-check=full to see details of leaked memory
   48: ==32448==
   49: ==32448== For counts of detected and suppressed errors, rerun with: -v
   50: ==32448== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
    1: ==32449== Memcheck, a memory error detector
    2: ==32449== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
    3: ==32449== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
    4: ==32449== Command: treefree -d
    5: ==32449==
    6: Command: treefree -d
    7: [0]0x5a24110->tree::tree(tree_ptr): parent=nullptr
    8: [1]0x5a24220->tree::tree(tree_ptr): parent=[0]0x5a24110
    9: [2]0x5a24410->tree::tree(tree_ptr): parent=[0]0x5a24110
   10: [3]0x5a24660->tree::tree(tree_ptr): parent=[2]0x5a24410
   11: [4]0x5a248b0->tree::tree(tree_ptr): parent=[2]0x5a24410
   12: [5]0x5a24b00->tree::tree(tree_ptr): parent=[2]0x5a24410
   13: [6]0x5a24cf0->tree::tree(tree_ptr): parent=[1]0x5a24220
   14: [7]0x5a24f40->tree::tree(tree_ptr): parent=[1]0x5a24220
   15: [8]0x5a25190->tree::tree(tree_ptr): parent=[1]0x5a24220
   16: [seq]address: key -> value (use count)
   17: void tree::print(size_t): [0]0x5a24110: ".." -> [0]0x5a24110 (5)
   18: void tree::print(size_t): [0]0x5a24110: "bar" -> [2]0x5a24410 (5)
   19: void tree::print(size_t):    [2]0x5a24410: ".." -> [0]0x5a24110 (5)
   20: void tree::print(size_t):    [2]0x5a24410: "quuux" -> [5]0x5a24b00 (2)
   21: void tree::print(size_t):       [5]0x5a24b00: ".." -> [2]0x5a24410 (6)
   22: void tree::print(size_t):    [2]0x5a24410: "quux" -> [4]0x5a248b0 (2)
   23: void tree::print(size_t):       [4]0x5a248b0: ".." -> [2]0x5a24410 (6)
   24: void tree::print(size_t):    [2]0x5a24410: "qux" -> [3]0x5a24660 (2)
   25: void tree::print(size_t):       [3]0x5a24660: ".." -> [2]0x5a24410 (6)
   26: void tree::print(size_t): [0]0x5a24110: "foo" -> [1]0x5a24220 (5)
   27: void tree::print(size_t):    [1]0x5a24220: ".." -> [0]0x5a24110 (5)
   28: void tree::print(size_t):    [1]0x5a24220: "quuux" -> [8]0x5a25190 (2)
   29: void tree::print(size_t):       [8]0x5a25190: ".." -> [1]0x5a24220 (6)
   30: void tree::print(size_t):    [1]0x5a24220: "quux" -> [7]0x5a24f40 (2)
   31: void tree::print(size_t):       [7]0x5a24f40: ".." -> [1]0x5a24220 (6)
   32: void tree::print(size_t):    [1]0x5a24220: "qux" -> [6]0x5a24cf0 (2)
   33: void tree::print(size_t):       [6]0x5a24cf0: ".." -> [1]0x5a24220 (6)
   34: void tree::disown(size_t): [0]0x5a24110
   35: void tree::disown(size_t):    [2]0x5a24410
   36: void tree::disown(size_t):       [5]0x5a24b00
   37: void tree::disown(size_t):       [4]0x5a248b0
   38: void tree::disown(size_t):       [3]0x5a24660
   39: void tree::disown(size_t):    [1]0x5a24220
   40: void tree::disown(size_t):       [8]0x5a25190
   41: void tree::disown(size_t):       [7]0x5a24f40
   42: void tree::disown(size_t):       [6]0x5a24cf0
   43: [0]0x5a24110->tree::~tree(): bar foo
   44: [1]0x5a24220->tree::~tree(): quuux quux qux
   45: [6]0x5a24cf0->tree::~tree():
   46: [7]0x5a24f40->tree::~tree():
   47: [8]0x5a25190->tree::~tree():
   48: [2]0x5a24410->tree::~tree(): quuux quux qux
   49: [3]0x5a24660->tree::~tree():
   50: [4]0x5a248b0->tree::~tree():
   51: [5]0x5a24b00->tree::~tree():
   52: ==32449==
   53: ==32449== HEAP SUMMARY:
   54: ==32449==     in use at exit: 0 bytes in 0 blocks
   55: ==32449==   total heap usage: 40 allocs, 40 frees, 2,002 bytes allocated
   56: ==32449==
```

```
57: ==32449== All heap blocks were freed -- no leaks are possible
58: ==32449==
59: ==32449== For counts of detected and suppressed errors, rerun with: -v
60: ==32449== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```