```
// $Id: treefree.cpp,v 1.75 2019-04-22 15:56:04-07 - - $

// Shared_ptrs use reference counting in order to automatically
// free objects, but that does not work for cyclic data structures.
// This illustrates how to avoid the problem.

#include <iomanip>
#include <iostream>
#include <map>
#include <memory>
using namespace std;

////////////////////////////////////////////////////////////////
// tree.h
////////////////////////////////////////////////////////////////

class tree;
using tree_ptr = shared_ptr<tree>;
using tree_dir = map<string,tree_ptr>;
using tree_itor = tree_dir::iterator;

class tree {
      friend ostream& operator<< (ostream&, const tree*);
   private:
      static size_t next_seq;
      size_t seq;
      tree_dir data;
      void print (size_t);
      void disown (size_t);
   public:
      static const string PARENT;
      static tree_ptr make_root();
      static tree_ptr make (tree_ptr ptr);
      explicit tree (tree_ptr parent);
      ~tree();
      void emplace (const tree_dir::key_type&,
                    const tree_dir::mapped_type&);
      const tree_itor begin() { return data.begin(); }
      const tree_itor end() { return data.end(); }
      void print() { print (0); }
      void disown() { disown (0); }
};

////////////////////////////////////////////////////////////////
// tree.cpp
////////////////////////////////////////////////////////////////

size_t tree::next_seq {0};
const string tree::PARENT = "..";

```

```

ostream& operator<< (ostream& out, const tree* ptr) {
   if (ptr == nullptr) return out << "nullptr";
                  else return out << "[" << ptr->seq << "]"
                                  << static_cast<const void*> (ptr);
}

tree::tree (tree_ptr parent): seq(next_seq++), data({{PARENT,parent}}) {
   cout << this << "->" << __PRETTY_FUNCTION__
        << "(" << parent << ")" << endl;
}

tree::~tree() {
   cout << this << "->" << __PRETTY_FUNCTION__ << "()" << endl;
}

void tree::emplace (const tree_dir::key_type& key,
                    const tree_dir::mapped_type& value) {
   data.emplace (key, value);
}

void tree::disown (size_t depth) {
   cout << __PRETTY_FUNCTION__ << ": "
        << setw (depth * 3) << "" << this << endl;
   data.erase (PARENT);
   for (auto n: data) n.second->disown (depth + 1);
}

// Depth-first pre-order traversal.
void tree::print (size_t depth) {
   for (const auto itor: data) {
      cout << __PRETTY_FUNCTION__ << ": "
           << setw (depth * 3) << "" << this
           << ": \"" << itor.first << "\" -> " << itor.second
           << " (" << itor.second.use_count() << ")" << endl;
      if (itor.first != PARENT and itor.second != nullptr) {
         itor.second->print (depth + 1);
      }
   }
}

tree_ptr tree::make_root() {
   tree_ptr ptr = make_shared<tree> (nullptr);
   ptr->data[PARENT] = ptr;
   return ptr;
}

tree_ptr tree::make (tree_ptr parent) {
   if (parent == nullptr) throw logic_error ("tree::make(nullptr)");
   return make_shared<tree> (parent);
}

```

```

////////////////////////////////////////////////////////////////////
// main.cpp
////////////////////////////////////////////////////////////////////

int main (int argc, char** argv) {
    (void) argc;
    (void) argv;
   shared_ptr<tree> root = tree::make_root();
   root->emplace ("foo", tree::make (root));
   root->emplace ("bar", tree::make (root));
   for (auto itor: *root) {
      if (itor.first == tree::PARENT) continue;
      for (int count = 0; count < 3; ++count) {
         string quux = "qux";
         quux.insert (1, count, 'u');
         itor.second->emplace (quux, tree::make (itor.second));
      }
   }
   cout << "[seq]address: key -> value (use count)" << endl;
   root->print();
   root->disown();
   return 0;
}

//TEST// alias grind='valgrind --leak-check=full --show-reachable=yes'
//TEST// grind treefree >treefree.out 2>treefree.ground
//TEST// mkpspdf treefree.ps treefree.cpp* treefree.out treefree.ground

```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting treefree.cpp
checksource treefree.cpp
ident treefree.cpp
treefree.cpp:
     $Id: treefree.cpp,v 1.75 2019-04-22 15:56:04-07 - - $
cpplint.py.perl treefree.cpp
Done processing treefree.cpp
g++ -Wall -Wextra -Wpedantic -Wshadow -fdiagnostics-color=never -std=gnu
++2a -Wold-style-cast -g -O0 treefree.cpp -o treefree -lm
rm -f treefree.o
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished treefree.cpp
```

```
[0]0x5a240b0->tree::tree(tree_ptr)(nullptr)
[1]0x5a241c0->tree::tree(tree_ptr)([0]0x5a240b0)
[2]0x5a243b0->tree::tree(tree_ptr)([0]0x5a240b0)
[3]0x5a24600->tree::tree(tree_ptr)([2]0x5a243b0)
[4]0x5a24850->tree::tree(tree_ptr)([2]0x5a243b0)
[5]0x5a24aa0->tree::tree(tree_ptr)([2]0x5a243b0)
[6]0x5a24c90->tree::tree(tree_ptr)([1]0x5a241c0)
[7]0x5a24ee0->tree::tree(tree_ptr)([1]0x5a241c0)
[8]0x5a25130->tree::tree(tree_ptr)([1]0x5a241c0)
[seq]address: key -> value (use count)
void tree::print(size_t): [0]0x5a240b0: ".." -> [0]0x5a240b0 (5)
void tree::print(size_t): [0]0x5a240b0: "bar" -> [2]0x5a243b0 (5)
void tree::print(size_t):    [2]0x5a243b0: ".." -> [0]0x5a240b0 (5)
void tree::print(size_t):    [2]0x5a243b0: "quuux" -> [5]0x5a24aa0 (2)
void tree::print(size_t):       [5]0x5a24aa0: ".." -> [2]0x5a243b0 (6)
void tree::print(size_t):    [2]0x5a243b0: "quux" -> [4]0x5a24850 (2)
void tree::print(size_t):       [4]0x5a24850: ".." -> [2]0x5a243b0 (6)
void tree::print(size_t):    [2]0x5a243b0: "qux" -> [3]0x5a24600 (2)
void tree::print(size_t):       [3]0x5a24600: ".." -> [2]0x5a243b0 (6)
void tree::print(size_t): [0]0x5a240b0: "foo" -> [1]0x5a241c0 (5)
void tree::print(size_t):    [1]0x5a241c0: ".." -> [0]0x5a240b0 (5)
void tree::print(size_t):    [1]0x5a241c0: "quuux" -> [8]0x5a25130 (2)
void tree::print(size_t):       [8]0x5a25130: ".." -> [1]0x5a241c0 (6)
void tree::print(size_t):    [1]0x5a241c0: "quux" -> [7]0x5a24ee0 (2)
void tree::print(size_t):       [7]0x5a24ee0: ".." -> [1]0x5a241c0 (6)
void tree::print(size_t):    [1]0x5a241c0: "qux" -> [6]0x5a24c90 (2)
void tree::print(size_t):       [6]0x5a24c90: ".." -> [1]0x5a241c0 (6)
void tree::disown(size_t): [0]0x5a240b0
void tree::disown(size_t):    [2]0x5a243b0
void tree::disown(size_t):       [5]0x5a24aa0
void tree::disown(size_t):       [4]0x5a24850
void tree::disown(size_t):       [3]0x5a24600
void tree::disown(size_t):    [1]0x5a241c0
void tree::disown(size_t):       [8]0x5a25130
void tree::disown(size_t):       [7]0x5a24ee0
void tree::disown(size_t):       [6]0x5a24c90
[0]0x5a240b0->tree::~tree()()
[1]0x5a241c0->tree::~tree()()
[6]0x5a24c90->tree::~tree()()
[7]0x5a24ee0->tree::~tree()()
[8]0x5a25130->tree::~tree()()
[2]0x5a243b0->tree::~tree()()
[3]0x5a24600->tree::~tree()()
[4]0x5a24850->tree::~tree()()
[5]0x5a24aa0->tree::~tree()()
```

```
==10244== Memcheck, a memory error detector
==10244== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10244== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==10244== Command: treefree
==10244==
==10244==
==10244== HEAP SUMMARY:
==10244==     in use at exit: 0 bytes in 0 blocks
==10244==   total heap usage: 39 allocs, 39 frees, 1,975 bytes allocated
==10244==
==10244== All heap blocks were freed -- no leaks are possible
==10244==
==10244== For counts of detected and suppressed errors, rerun with: -v
==10244== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```