```
 1: // $Id: shared_ptrs.cpp,v 1.63 2022-01-14 20:43:06-08 - - $
 2:
 3: #include <iostream>
 4: #include <memory>
 5: #include <sstream>
 6: #include <string>
 7: using namespace std;
 8:
 9: // Illustrate use of shared pointers and copying.
10:
11: const string indent (5, ' ');
12:
13: #define SHOWBOX { cout << indent << __PRETTY_FUNCTION__ << ": " \
14:                         << *this << endl; }
15:
16: using sbox_ptr = shared_ptr<struct sbox>;
17:
18: class sbox {
19:    friend ostream& operator<< (ostream&, const sbox&);
20:    private:
21:       string value {"<EMPTY>"};
22:    public:
23:       sbox() {SHOWBOX}
24:       sbox (const sbox& that): value(that.value) {SHOWBOX} // copier
25:       sbox (sbox&& that): value(that.value) {SHOWBOX}      // mover
26:       sbox& operator= (const sbox& that); // copier
27:       sbox& operator= (sbox&& that);      // mover
28:       ~sbox() {SHOWBOX}
29:       sbox (const string& val): value(val) {SHOWBOX}
30:       const string& operator*() const { return value; }
31: };
32:
33: sbox& sbox::operator= (const sbox& that) {
34:    if (this != &that) value = that.value;
35:    SHOWBOX;
36:    return *this;
37: }
38:
39: sbox& sbox::operator= (sbox&& that) {
40:    if (this != &that) value = that.value;
41:    SHOWBOX;
42:    return *this;
43: }
44:
45: ostream& operator<< (ostream& out, const sbox& box) {
46:    return out << &box << ":sbox(\"" << box.value << "\")";
47: }
48:
```

```
49:
50: template <typename Type>
51: ostream& operator<< (ostream& out, shared_ptr<Type> ptr) {
52:    return out << "{" << ptr.get() << "," << ptr.use_count() << "}";
53: }
54:
55: #define LINE "[" << __LINE__ << "] "
56: #define SHOW(STMT) cout << LINE << #STMT << endl; STMT;
57:
58: void show_ptr (const sbox_ptr& ptr) {
59:    cout << indent << ptr << " -> ";
60:    if (ptr) cout << *ptr; else cout << "nullptr";
61:    cout << endl;
62: }
63:
64: int main() {
65:    SHOW( sbox_ptr junk {make_shared<sbox> (":junk:")}; )
66:    SHOW( junk = nullptr; )
67:    SHOW( sbox_ptr a {make_shared<sbox>()}; )
68:    SHOW( show_ptr(a); )
69:    SHOW( sbox_ptr b {make_shared<sbox> ("foobar")}; )
70:    SHOW( auto single {make_shared<sbox> ("single")}; )
71:    SHOW( show_ptr(single); )
72:    SHOW( show_ptr(b); )
73:    SHOW( a = b; )
74:    SHOW( show_ptr(a); )
75:    SHOW( show_ptr(b); )
76:    SHOW( sbox_ptr c {a}; )
77:    SHOW( show_ptr(c); )
78:    SHOW( b = nullptr; )
79:    SHOW( show_ptr(b); )
80:    SHOW( show_ptr(a); )
81:    SHOW( return 0; )
82: }
83:
84: //TEST// valgrind shared_ptrs >shared_ptrs.out 2>&1
85: //TEST// mkpspdf shared_ptrs.ps shared_ptrs.cpp shared_ptrs.out
86:
```

```
    1: ==15738== Memcheck, a memory error detector
    2: ==15738== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
    3: ==15738== Using Valgrind-3.17.0 and LibVEX; rerun with -h for copyright
info
    4: ==15738== Command: shared_ptrs
    5: ==15738==
    6: [65] sbox_ptr junk {make_shared<sbox> (":junk:")};
    7:      sbox::sbox(const string&): 0x5c450b0:sbox(":junk:")
    8: [66] junk = nullptr;
    9:      sbox::~sbox(): 0x5c450b0:sbox(":junk:")
   10: [67] sbox_ptr a {make_shared<sbox>()};
   11:      sbox::sbox(): 0x5c45170:sbox("<EMPTY>")
   12: [68] show_ptr(a);
   13:      {0x5c45170,2} -> 0x5c45170:sbox("<EMPTY>")
   14: [69] sbox_ptr b {make_shared<sbox> ("foobar")};
   15:      sbox::sbox(const string&): 0x5c45230:sbox("foobar")
   16: [70] auto single {make_shared<sbox> ("single")};
   17:      sbox::sbox(const string&): 0x5c452f0:sbox("single")
   18: [71] show_ptr(single);
   19:      {0x5c452f0,2} -> 0x5c452f0:sbox("single")
   20: [72] show_ptr(b);
   21:      {0x5c45230,2} -> 0x5c45230:sbox("foobar")
   22: [73] a = b;
   23:      sbox::~sbox(): 0x5c45170:sbox("<EMPTY>")
   24: [74] show_ptr(a);
   25:      {0x5c45230,3} -> 0x5c45230:sbox("foobar")
   26: [75] show_ptr(b);
   27:      {0x5c45230,3} -> 0x5c45230:sbox("foobar")
   28: [76] sbox_ptr c {a};
   29: [77] show_ptr(c);
   30:      {0x5c45230,4} -> 0x5c45230:sbox("foobar")
   31: [78] b = nullptr;
   32: [79] show_ptr(b);
   33:      {0,0} -> nullptr
   34: [80] show_ptr(a);
   35:      {0x5c45230,3} -> 0x5c45230:sbox("foobar")
   36: [81] return 0;
   37:      sbox::~sbox(): 0x5c452f0:sbox("single")
   38:      sbox::~sbox(): 0x5c45230:sbox("foobar")
   39: ==15738==
   40: ==15738== HEAP SUMMARY:
   41: ==15738==     in use at exit: 0 bytes in 0 blocks
   42: ==15738==   total heap usage: 9 allocs, 9 frees, 251 bytes allocated
   43: ==15738==
   44: ==15738== All heap blocks were freed -- no leaks are possible
   45: ==15738==
   46: ==15738== For lists of detected and suppressed errors, rerun with: -s
   47: ==15738== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```