```
 1: // $Id: inheritance2.cpp,v 1.19 2016-07-21 19:46:50-07 - - $
 2:
 3: //
 4: // Example using objects, with a base object and two derived objects.
 5: // Similar to inheritance2, but uses gcc demangler.
 6: //
 7:
 8: #include <iostream>
 9: #include <memory>
10: #include <typeinfo>
11: #include <vector>
12: using namespace std;
13:
14: #define LOG cout << "[" << __LINE__ << "]" \
15:                  << __PRETTY_FUNCTION__ << ": "
16:
17: #include <cxxabi.h>
18: template <typename type>
19: string demangle_typeid (const type& object) {
20:    const char* name = typeid(object).name();
21:    int status = 0;
22:    using deleter = void (*) (void*);
23:    unique_ptr<char,deleter> result {
24:       abi::__cxa_demangle (name, nullptr, nullptr, &status),
25:       std::free,
26:    };
27:    return status == 0 ? result.get() : name;
28: }
29:
```

```
30:
31: //////////////////////////////////////////////////////////////////
32: // class object
33: //////////////////////////////////////////////////////////////////
34:
35: class object {
36:    private:
37:       object (const object&) = delete;
38:       object& operator= (const object&) = delete;
39:       object (object&&) = delete;
40:       object& operator= (object&&) = delete;
41:       static unsigned next_id;;
42:    protected:
43:       const unsigned id;
44:       object(); // abstract class, so only derived can used ctor.
45:    public:
46:       virtual ~object(); // must be virtual
47:       virtual void print (ostream&) const;
48: };
49:
50: ostream& operator<< (ostream& out, const object& obj) {
51:    obj.print (out);
52:    return out;
53: }
54:
55: unsigned object::next_id = 0;
56:
57: object::object(): id(++next_id) {
58:    LOG << *this << endl;
59: }
60:
61: object::~object() {
62:    LOG << *this << endl;
63: }
64:
65: void object::print (ostream& out) const {
66:    out << "[" << static_cast<const void *const> (this) << "->"
67:        << demangle_typeid(*this) << "] id=" << id << ": ";
68: }
69:
```

```
 70:
 71: //////////////////////////////////////////////////////////////////
 72: // class square
 73: //////////////////////////////////////////////////////////////////
 74:
 75: class square: public object {
 76:    private:
 77:       size_t width;
 78:    public:
 79:       explicit square (size_t width = 0);
 80:       virtual ~square();
 81:       virtual void print (ostream&) const;
 82: };
 83:
 84: square::square (size_t width): width(width) {
 85:    LOG << *this << endl;
 86: }
 87:
 88: square::~square() {
 89:    LOG << *this << endl;
 90: }
 91:
 92: void square::print (ostream& out) const {
 93:    this->object::print (out);
 94:    out << "width=" << width;
 95: }
 96:
 97: //////////////////////////////////////////////////////////////////
 98: // class circle
 99: //////////////////////////////////////////////////////////////////
100:
101: class circle: public object {
102:    private:
103:       size_t diam;
104:    public:
105:       explicit circle (size_t diam = 0);
106:       virtual ~circle();
107:       virtual void print (ostream&) const;
108: };
109:
110: circle::circle (size_t diam): diam(diam) {
111:    LOG << *this << endl;
112: }
113:
114: circle::~circle() {
115:    LOG << *this << endl;
116: }
117:
118: void circle::print (ostream& out) const {
119:    this->object::print (out);
120:    out << "diam=" << diam;
121: }
122:
123:
```

```
124:
125: //////////////////////////////////////////////////////////////////
126: // main
127: //////////////////////////////////////////////////////////////////
128:
129: int main() {
130:    LOG << "sizeof (object) = " << sizeof (object) << endl;
131:    LOG << "sizeof (square) = " << sizeof (square) << endl;
132:    LOG << "sizeof (circle) = " << sizeof (circle) << endl;
133:
134:    vector<shared_ptr<object>> vec;
135:    // ERROR: v.push_back (new object());
136:    // ERROR: object o;
137:    vec.push_back (make_shared<circle> (  ));
138:    vec.push_back (make_shared<circle> (10));
139:    vec.push_back (make_shared<square> (  ));
140:    vec.push_back (make_shared<square> ( 5));
141:    vec.push_back (make_shared<square> ( 8));
142:    cout << endl;
143:
144:    for (const auto& ptr: vec) {
145:       LOG << "Object: " << *ptr << endl;
146:    }
147:    cout << endl;
148:
149:    LOG << "return 0" << endl;
150:    return 0;
151: }
152:
153: /*
154: //TEST// valgrind --leak-check=full --show-reachable=yes \
155: //TEST//       inheritance2 >inheritance2.out 2>&1
156: //TEST// mkpspdf inheritance2.ps inheritance2.cpp* inheritance2.out*
157: */
158:
```

```
    1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting inheritance2.cpp
    2: inheritance2.cpp: $Id: inheritance2.cpp,v 1.19 2016-07-21 19:46:50-07 -
- $
    3: g++ -g -O0 -std=gnu++14 -rdynamic -Wall -Wextra -Wold-style-cast inherit
ance2.cpp -o inheritance2 -lglut -lGLU -lGL -lX11 -lrt -lm
    4: cpplint.py.perl inheritance2.cpp
    5: Done processing inheritance2.cpp
    6: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished inheritance2.cpp
```

```
    1: ==28710== Memcheck, a memory error detector
    2: ==28710== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al
.
    3: ==28710== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright
info
    4: ==28710== Command: inheritance2
    5: ==28710==
    6: [130]int main(): sizeof (object) = 16
    7: [131]int main(): sizeof (square) = 24
    8: [132]int main(): sizeof (circle) = 24
    9: [58]object::object(): [0x9c9b0a0->object] id=1:
   10: [111]circle::circle(size_t): [0x9c9b0a0->circle] id=1: diam=0
   11: [58]object::object(): [0x9c9b2c0->object] id=2:
   12: [111]circle::circle(size_t): [0x9c9b2c0->circle] id=2: diam=10
   13: [58]object::object(): [0x9c9b4f0->object] id=3:
   14: [85]square::square(size_t): [0x9c9b4f0->square] id=3: width=0
   15: [58]object::object(): [0x9c9b740->object] id=4:
   16: [85]square::square(size_t): [0x9c9b740->square] id=4: width=5
   17: [58]object::object(): [0x9c9b910->object] id=5:
   18: [85]square::square(size_t): [0x9c9b910->square] id=5: width=8
   19:
   20: [145]int main(): Object: [0x9c9b0a0->circle] id=1: diam=0
   21: [145]int main(): Object: [0x9c9b2c0->circle] id=2: diam=10
   22: [145]int main(): Object: [0x9c9b4f0->square] id=3: width=0
   23: [145]int main(): Object: [0x9c9b740->square] id=4: width=5
   24: [145]int main(): Object: [0x9c9b910->square] id=5: width=8
   25:
   26: [149]int main(): return 0
   27: [115]virtual circle::~circle(): [0x9c9b0a0->circle] id=1: diam=0
   28: [62]virtual object::~object(): [0x9c9b0a0->object] id=1:
   29: [115]virtual circle::~circle(): [0x9c9b2c0->circle] id=2: diam=10
   30: [62]virtual object::~object(): [0x9c9b2c0->object] id=2:
   31: [89]virtual square::~square(): [0x9c9b4f0->square] id=3: width=0
   32: [62]virtual object::~object(): [0x9c9b4f0->object] id=3:
   33: [89]virtual square::~square(): [0x9c9b740->square] id=4: width=5
   34: [62]virtual object::~object(): [0x9c9b740->object] id=4:
   35: [89]virtual square::~square(): [0x9c9b910->square] id=5: width=8
   36: [62]virtual object::~object(): [0x9c9b910->object] id=5:
   37: ==28710==
   38: ==28710== HEAP SUMMARY:
   39: ==28710==     in use at exit: 0 bytes in 0 blocks
   40: ==28710==   total heap usage: 60 allocs, 60 frees, 1,431 bytes allocated
   41: ==28710==
   42: ==28710== All heap blocks were freed -- no leaks are possible
   43: ==28710==
   44: ==28710== For counts of detected and suppressed errors, rerun with: -v
   45: ==28710== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 1 from 1)
```