```
 1: // $Id: virtual-trace.cpp,v 1.21 2022-01-24 14:07:38-08 - - $
 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 <cmath>
 9: #include <iomanip>
10: #include <iostream>
11: #include <memory>
12: #include <typeinfo>
13: #include <vector>
14: using namespace std;
15:
16: #define LOG cout << "[" << __PRETTY_FUNCTION__ << "] "
17:
18: #include <cxxabi.h>
19: template <typename type>
20: string demangle_typeid (const type& object) {
21:    const char* name = typeid(object).name();
22:    int status = 0;
23:    unique_ptr<char,decltype(&std::free)> 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:       static size_t next_id;
38:    protected:
39:       const size_t id;
40:       object(); // abstract class, so only derived can used ctor.
41:    public:
42:       object (const object&) = delete;             // no copying
43:       object& operator= (const object&) = delete; // no copying
44:       virtual ~object(); // must be virtual
45:       virtual ostream& print (ostream&) const;
46:       virtual double area() const = 0;
47: };
48: size_t object::next_id = 0;
49:
50: ostream& operator<< (ostream& out, const object& obj) {
51:    return obj.print (out);
52: }
53:
54: object::object(): id(next_id++) {
55:    LOG << *this << endl;
56: }
57:
58: object::~object() {
59:    LOG << *this << endl;
60: }
61:
62: ostream& object::print (ostream& out) const {
63:    return out << static_cast<const void*> (this) << "->"
64:               << demangle_typeid(*this) << ":id=" << id;
65: }
66:
```

```
67:
68: //////////////////////////////////////////////////////////////////
69: // class square
70: //////////////////////////////////////////////////////////////////
71:
72: class square: public object {
73:    private:
74:       double width;
75:    public:
76:       explicit square (size_t width);
77:       virtual ~square();
78:       virtual ostream& print (ostream&) const override;
79:       virtual double area() const override;
80: };
81:
82: square::square (size_t width_): width(width_) {
83:    LOG << *this << endl;
84: }
85:
86: square::~square() {
87:    LOG << *this << endl;
88: }
89:
90: ostream& square::print (ostream& out) const {
91:    return this->object::print (out) << ", width=" << width;
92: }
93:
94: double square::area() const {
95:    return pow (width, 2.0);
96: }
97:
```

```
 98:
 99: //////////////////////////////////////////////////////////////////
100: // class circle
101: //////////////////////////////////////////////////////////////////
102:
103: class circle: public object {
104:    private:
105:       double radius;
106:    public:
107:       explicit circle (size_t radius);
108:       virtual ~circle();
109:       virtual ostream& print (ostream&) const override;
110:       virtual double area() const override;
111: };
112:
113: circle::circle (size_t radius_): radius(radius_) {
114:    LOG << *this << endl;
115: }
116:
117: circle::~circle() {
118:    LOG << *this << endl;
119: }
120:
121: ostream& circle::print (ostream& out) const {
122:    return this->object::print (out) << ", radius=" << radius;
123: }
124:
125: double circle::area() const {
126:    return M_PI * pow (radius, 2.0);
127: }
128:
```

```
129:
130: //////////////////////////////////////////////////////////////////
131: // main
132: //////////////////////////////////////////////////////////////////
133:
134: int main() {
135:
136:    vector<shared_ptr<object>> vec;
137:    LOG << "Before push_back ..." << endl;
138:    vec.push_back (make_shared<circle> ( 5));
139:    vec.push_back (make_shared<square> ( 5));
140:    vec.push_back (make_shared<circle> (10));
141:    vec.push_back (make_shared<square> (10));
142:    cout << endl;
143:
144:    LOG << "Before for first for loop ..." << endl;
145:    for (const auto& ptr: vec) {
146:       LOG << *ptr << " ...area=" << ptr->area() << endl;
147:    }
148:    cout << endl;
149:
150:    LOG << "Before pop_back for loop ..." << endl;
151:    vec.pop_back();
152:    vec.pop_back();
153:    cout << endl;
154:
155:    LOG << "Before return 0 ..." << endl;
156:    return 0;
157: }
158:
159: //TEST// valgrind virtual-trace >virtual-trace.out 2>&1
160: //TEST// mkpspdf virtual-trace.ps virtual-trace.cpp virtual-trace.out
161:
```

```
    1: ==10829== Memcheck, a memory error detector
    2: ==10829== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
    3: ==10829== Using Valgrind-3.17.0 and LibVEX; rerun with -h for copyright
info
    4: ==10829== Command: virtual-trace
    5: ==10829==
    6: [int main()] Before push_back ...
    7: [object::object()] 0x5c45050->object:id=0
    8: [circle::circle(size_t)] 0x5c45050->circle:id=0, radius=5
    9: [object::object()] 0x5c45270->object:id=1
   10: [square::square(size_t)] 0x5c45270->square:id=1, width=5
   11: [object::object()] 0x5c454a0->object:id=2
   12: [circle::circle(size_t)] 0x5c454a0->circle:id=2, radius=10
   13: [object::object()] 0x5c456f0->object:id=3
   14: [square::square(size_t)] 0x5c456f0->square:id=3, width=10
   15:
   16: [int main()] Before for first for loop ...
   17: [int main()] 0x5c45050->circle:id=0, radius=5 ...area=78.5398
   18: [int main()] 0x5c45270->square:id=1, width=5 ...area=25
   19: [int main()] 0x5c454a0->circle:id=2, radius=10 ...area=314.159
   20: [int main()] 0x5c456f0->square:id=3, width=10 ...area=100
   21:
   22: [int main()] Before pop_back for loop ...
   23: [virtual square::~square()] 0x5c456f0->square:id=3, width=10
   24: [virtual object::~object()] 0x5c456f0->object:id=3
   25: [virtual circle::~circle()] 0x5c454a0->circle:id=2, radius=10
   26: [virtual object::~object()] 0x5c454a0->object:id=2
   27:
   28: [int main()] Before return 0 ...
   29: [virtual circle::~circle()] 0x5c45050->circle:id=0, radius=5
   30: [virtual object::~object()] 0x5c45050->object:id=0
   31: [virtual square::~square()] 0x5c45270->square:id=1, width=5
   32: [virtual object::~object()] 0x5c45270->object:id=1
   33: ==10829==
   34: ==10829== HEAP SUMMARY:
   35: ==10829==     in use at exit: 0 bytes in 0 blocks
   36: ==10829==   total heap usage: 47 allocs, 47 frees, 1,052 bytes allocated
   37: ==10829==
   38: ==10829== All heap blocks were freed -- no leaks are possible
   39: ==10829==
   40: ==10829== For lists of detected and suppressed errors, rerun with: -s
   41: ==10829== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```