

```
1: // $Id: typeid.cpp,v 1.17 2015-02-10 17:53:15-08 - - $
2:
3: //
4: // Show a few classes with inheritance.
5: // All are inline for simplicity.
6: // Suppress copy ctor and operator= because of inheritance.
7: // Always make dtor virtual if any functions are virtual.
8: //
9:
10: #include <iostream>
11: #include <string>
12: #include <typeinfo>
13:
14: using namespace std;
15: #define TRACE(STMT) cout << "TRACE (" << #STMT << ")" << endl; STMT
16: #define SHOW cout << this << "->" << __func__ << endl;
17:
18: class base {
19:     private:
20:         string baseid;
21:         base (const base&) = delete;
22:         base& operator= (const base&) = delete;
23:     public:
24:         base(): baseid ("(base)") { SHOW }
25:         virtual ~base() { SHOW }
26:         virtual void print() { cout << baseid; }
27: };
28:
29: class extend: public base {
30:     private:
31:         string extendid;
32:     public:
33:         extend(): extendid ("(extend)") { SHOW }
34:         virtual ~extend() { SHOW }
35:         virtual void print() { base::print(); cout << "::" << extendid; }
36: };
37:
38: class other: public base {
39:     private:
40:         string otherid;
41:     public:
42:         other(): otherid ("(other)") { SHOW }
43:         virtual ~other() { SHOW }
44:         virtual void print() { base::print(); cout << "::" << otherid; }
45: };
46:
```

```
47:
48: int main() {
49:     TRACE (cout << sizeof (base) << endl;)
50:     TRACE (cout << sizeof (extend) << endl;)
51:     TRACE (cout << sizeof (other) << endl;)
52:     TRACE (base *bptr = new base();)
53:     TRACE (cout << "typeid *bptr = " << typeid (*bptr).name() << endl;)
54:     TRACE (bptr->print(); cout << endl;)
55:     TRACE (base *dptr = new extend();)
56:     TRACE (cout << "typeid *dptr = " << typeid (*dptr).name() << endl;)
57:     TRACE (dptr->print(); cout << endl;)
58:     TRACE (base *optr = new other();)
59:     TRACE (cout << "typeid *optr = " << typeid (*optr).name() << endl;)
60:     TRACE (optr->print(); cout << endl;)
61:     TRACE (delete optr;)
62:     TRACE (delete bptr;)
63:     TRACE (bptr = dptr;)
64:     TRACE (cout << "typeid *bptr = " << typeid (*bptr).name() << endl;)
65:     TRACE (bptr->print(); cout << endl;)
66:     TRACE (dptr = dynamic_cast<extend*> (bptr);)
67:     TRACE (cout << "typeid *dptr = " << typeid (*dptr).name() << endl;)
68:     TRACE (dptr->print(); cout << endl;)
69:     TRACE (extend dloc;)
70:     TRACE (cout << "typeid dloc = " << typeid (dloc).name() << endl;)
71:     TRACE (dloc.print(); cout << endl;)
72:     TRACE (delete dptr;)
73:     //error: 'base::base(const base&)' is private
74:     //Otherwise we would get slicing.
75:     TRACE (return 0;)
76: }
77:
78: //TEST// grind="valgrind --leak-check=full --show-reachable=yes"
79: //TEST// $grind typeid >typeid.out 2>&1
80: //TEST// mkpspdf typeid.ps typeid.cpp* typeid.out*
81:
```

lut

```
1: ==8638== Memcheck, a memory error detector
2: ==8638== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
3: ==8638== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright in
fo
4: ==8638== Command: typeid
5: ==8638==
6: TRACE (cout << sizeof (base) << endl;)
7: 16
8: TRACE (cout << sizeof (extend) << endl;)
9: 24
10: TRACE (cout << sizeof (other) << endl;)
11: 24
12: TRACE (base *bptr = new base();)
13: 0x4e7d090->base
14: TRACE (cout << "typeid *bptr = " << typeid (*bptr).name() << endl;)
15: typeid *bptr = 4base
16: TRACE (bptr->print(); cout << endl;)
17: (base)
18: TRACE (base *dptr = new extend();)
19: 0x4e7d140->base
20: 0x4e7d140->extend
21: TRACE (cout << "typeid *dptr = " << typeid (*dptr).name() << endl;)
22: typeid *dptr = 6extend
23: TRACE (dptr->print(); cout << endl;)
24: (base)::(extend)
25: TRACE (base *optr = new other();)
26: 0x4e7d270->base
27: 0x4e7d270->other
28: TRACE (cout << "typeid *optr = " << typeid (*optr).name() << endl;)
29: typeid *optr = 5other
30: TRACE (optr->print(); cout << endl;)
31: (base)::(other)
32: TRACE (delete optr;)
33: 0x4e7d270->~other
34: 0x4e7d270->~base
35: TRACE (delete bptr;)
36: 0x4e7d090->~base
37: TRACE (bptr = dptr;)
38: TRACE (cout << "typeid *bptr = " << typeid (*bptr).name() << endl;)
39: typeid *bptr = 6extend
40: TRACE (bptr->print(); cout << endl;)
41: (base)::(extend)
42: TRACE (dptr = dynamic_cast<extend*> (bptr);)
43: TRACE (cout << "typeid *dptr = " << typeid (*dptr).name() << endl;)
44: typeid *dptr = 6extend
45: TRACE (dptr->print(); cout << endl;)
46: (base)::(extend)
47: TRACE (extend dloc;)
48: 0xffefff940->base
49: 0xffefff940->extend
50: TRACE (cout << "typeid dloc = " << typeid (dloc).name() << endl;)
51: typeid dloc = 6extend
52: TRACE (dloc.print(); cout << endl;)
53: (base)::(extend)
54: TRACE (delete dptr;)
55: 0x4e7d140->~extend
56: 0x4e7d140->~base
57: TRACE (return 0;)
```

```
58: 0xfffffff940->~extend
59: 0xfffffff940->~base
60: ==8638==
61: ==8638== HEAP SUMMARY:
62: ==8638==      in use at exit: 0 bytes in 0 blocks
63: ==8638==    total heap usage: 11 allocs, 11 frees, 295 bytes allocated
64: ==8638==
65: ==8638== All heap blocks were freed -- no leaks are possible
66: ==8638==
67: ==8638== For counts of detected and suppressed errors, rerun with: -v
68: ==8638== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```