```
// $Id: typeid.cpp,v 1.18 2016-07-20 16:27:47-07 - - $

//
// Show a few classes with inheritance.
// All are inline for simplicity.
// Suppress copy ctor and operator= because of inheritance.
// Always make dtor virtual if any functions are virtual.
//

#include <iostream>
#include <string>
#include <typeinfo>

using namespace std;
#define TRACE(STMT) cout << "TRACE (" << #STMT << ")" << endl; STMT
#define SHOW cout << this << "->" << __PRETTY_FUNCTION__ << endl;

class base {
   private:
      string baseid;
      base (const base&) = delete;
      base& operator= (const base&) = delete;
   public:
      base(): baseid ("(base)") { SHOW }
      virtual ~base() { SHOW }
      virtual void print() { cout << baseid; }
};

class extend: public base {
   private:
      string extendid;
   public:
      extend(): extendid ("(extend)") { SHOW }
      virtual ~extend() { SHOW }
      virtual void print() { base::print(); cout << "::" << extendid; }
};

class other: public base {
   private:
      string otherid;
   public:
      other(): otherid ("(other)") { SHOW }
      virtual ~other() { SHOW }
      virtual void print() { base::print(); cout << "::" << otherid; }
};

```

```

int main() {
   TRACE (cout << sizeof (base) << endl;)
   TRACE (cout << sizeof (extend) << endl;)
   TRACE (cout << sizeof (other) << endl;)
   TRACE (base *bptr = new base();)
   TRACE (cout << "typeid *bptr = " << typeid (*bptr).name() << endl;)
   TRACE (bptr->print(); cout << endl;)
   TRACE (base *dptr = new extend();)
   TRACE (cout << "typeid *dptr = " << typeid (*dptr).name() << endl;)
   TRACE (dptr->print(); cout << endl;)
   TRACE (base *optr = new other();)
   TRACE (cout << "typeid *optr = " << typeid (*optr).name() << endl;)
   TRACE (optr->print(); cout << endl;)
   TRACE (delete optr;)
   TRACE (delete bptr;)
   TRACE (bptr = dptr;)
   TRACE (cout << "typeid *bptr = " << typeid (*bptr).name() << endl;)
   TRACE (bptr->print(); cout << endl;)
   TRACE (dptr = dynamic_cast<extend *> (bptr);)
   TRACE (cout << "typeid *dptr = " << typeid (*dptr).name() << endl;)
   TRACE (dptr->print(); cout << endl;)
   TRACE (extend dloc;)
   TRACE (cout << "typeid dloc = " << typeid (dloc).name() << endl;)
   TRACE (dloc.print(); cout << endl;)
   TRACE (delete dptr;)
   //error: 'base::base(const base&)' is private
   //Otherwise we would get slicing.
   TRACE (return 0;)
}

//TEST// grind="valgrind --leak-check=full --show-reachable=yes"
//TEST// $grind typeid >typeid.out 2>&1
//TEST// mkpspdf typeid.ps typeid.cpp* typeid.out*

```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting typeid.cpp
typeid.cpp:
     $Id: typeid.cpp,v 1.18 2016-07-20 16:27:47-07 - - $
g++ -g -O0 -Wall -Wextra -rdynamic -std=gnu++14 typeid.cpp
        -o typeid -lglut -lGLU -lGL -lX11 -lrt -lm
rm -f typeid.o
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished typeid.cpp
```

```
==24877== Memcheck, a memory error detector
==24877== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al
.
==24877== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright
info
==24877== Command: typeid
==24877==
TRACE (cout << sizeof (base) << endl;)
16
TRACE (cout << sizeof (extend) << endl;)
24
TRACE (cout << sizeof (other) << endl;)
24
TRACE (base *bptr = new base();)
0x9c9b090->base::base()
TRACE (cout << "typeid *bptr = " << typeid (*bptr).name() << endl;)
typeid *bptr = 4base
TRACE (bptr->print(); cout << endl;)
(base)
TRACE (base *dptr = new extend();)
0x9c9b140->base::base()
0x9c9b140->extend::extend()
TRACE (cout << "typeid *dptr = " << typeid (*dptr).name() << endl;)
typeid *dptr = 6extend
TRACE (dptr->print(); cout << endl;)
(base)::(extend)
TRACE (base *optr = new other();)
0x9c9b270->base::base()
0x9c9b270->other::other()
TRACE (cout << "typeid *optr = " << typeid (*optr).name() << endl;)
typeid *optr = 5other
TRACE (optr->print(); cout << endl;)
(base)::(other)
TRACE (delete optr;)
0x9c9b270->virtual other::~other()
0x9c9b270->virtual base::~base()
TRACE (delete bptr;)
0x9c9b090->virtual base::~base()
TRACE (bptr = dptr;)
TRACE (cout << "typeid *bptr = " << typeid (*bptr).name() << endl;)
typeid *bptr = 6extend
TRACE (bptr->print(); cout << endl;)
(base)::(extend)
TRACE (dptr = dynamic_cast<extend *> (bptr);)
TRACE (cout << "typeid *dptr = " << typeid (*dptr).name() << endl;)
typeid *dptr = 6extend
TRACE (dptr->print(); cout << endl;)
(base)::(extend)
TRACE (extend dloc;)
0xffefff510->base::base()
0xffefff510->extend::extend()
TRACE (cout << "typeid dloc = " << typeid (dloc).name() << endl;)
typeid dloc = 6extend
TRACE (dloc.print(); cout << endl;)
(base)::(extend)
TRACE (delete dptr;)
0x9c9b140->virtual extend::~extend()
0x9c9b140->virtual base::~base()
```

```
TRACE (return 0;)
0xffefff510->virtual extend::~extend()
0xffefff510->virtual base::~base()
==24877==
==24877== HEAP SUMMARY:
==24877==     in use at exit: 0 bytes in 0 blocks
==24877==   total heap usage: 11 allocs, 11 frees, 302 bytes allocated
==24877==
==24877== All heap blocks were freed -- no leaks are possible
==24877==
==24877== For counts of detected and suppressed errors, rerun with: -v
==24877== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 1 from 1)
```