601.220 Intermediate Programming

Virtual destructors

Virtual Destructors

```
// virt_dtor.h
1   class Base {
2   public:
3       Base() : base_memory(new char[1000]) { }
4       ~Base() { delete[] base_memory; }
5
6   private:
7       char *base_memory;
8   };
9
10  class Derived : public Base {
11  public:
12      Derived() : Base(), derived_memory(new char[1000]) { }
13      ~Derived() { delete[] derived_memory; }
14
15  private:
16      char *derived_memory;
17  };
```

Virtual Destructors

```
// virt_dtor.cpp
1   #include "virt_dtor.h"
2
3   int main() {
4       // Note use of base-class pointer
5       Base *obj = new Derived();
6       delete obj; // calls what destructor(s)?
7       return 0;
8   }
```

`new Derived()` calls `Derived` default constructor, which in turn calls `Base` default constructor; that's good (both memories are allocated)

But which destructor is called?

- Destructor is not `virtual`
- Does that mean ˜Base is called but not ˜Derived?

Virtual Destructors

```
$ g++ -o virt_dtor virt_dtor.cpp -std=c++11 -pedantic -Wall -Wextra

$ valgrind --leak-check=full ./virt_dtor

==1202== Memcheck, a memory error detector
==1202== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==1202== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==1202== Command: ./virt_dtor
==1202==
==1202== error calling PR_SET_PTRACER, vgdb might block
==1202==
==1202== HEAP SUMMARY:
==1202==     in use at exit: 73,704 bytes in 2 blocks
==1202==   total heap usage: 4 allocs, 2 frees, 74,720 bytes allocated
==1202==
==1202== 1,000 bytes in 1 blocks are definitely lost in loss record 1 of 2
==1202==    at 0x4C2E80F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-lin
==1202==    by 0x400856: Derived::Derived() (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermmedia
==1202==    by 0x4007A1: main (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermmediate Programming
==1202==
==1202== LEAK SUMMARY:
==1202==    definitely lost: 1,000 bytes in 1 blocks
==1202==    indirectly lost: 0 bytes in 0 blocks
==1202==      possibly lost: 0 bytes in 0 blocks
==1202==    still reachable: 72,704 bytes in 1 blocks
==1202==         suppressed: 0 bytes in 0 blocks
==1202== Reachable blocks (those to which a pointer was found) are not shown.
==1202== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==1202==
==1202== For counts of detected and suppressed errors, rerun with: -v
==1202== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Virtual Destructors

```
// virt_dtor2.h
1   class Base {
2   public:
3       Base() : base_memory(new char[1000]) { }
4       virtual ~Base() { delete[] base_memory; }
5
6   private:
7       char *base_memory;
8   };
9
10  class Derived : public Base {
11  public:
12      Derived() : Base(), derived_memory(new char[1000]) { }
13      virtual ~Derived() { delete[] derived_memory; }
14
15  private:
16      char *derived_memory;
17  };
```

Virtual Destructors

```
// virt_dtor2.cpp
1  #include "virt_dtor2.h"
2
3  int main() {
4      // Note use of base-class pointer
5      Base *obj = new Derived();
6      delete obj; // calls what destructor(s)?
7      return 0;
8  }
```

This should fix the problem. Thanks to dynamic binding,
delete obj calls ~Derived, which in turn calls ~Base

Recall: derived-class destructor always implicitly calls base=class
destructor

Virtual Destructors

```
$ g++ -o virt_dtor2 virt_dtor2.cpp -std=c++11 -pedantic -Wall -Wextra

$ valgrind --leak-check=full ./virt_dtor2

==1212== Memcheck, a memory error detector
==1212== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==1212== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==1212== Command: ./virt_dtor2
==1212==
==1212== error calling PR_SET_PTRACER, vgdb might block
==1212==
==1212== HEAP SUMMARY:
==1212==     in use at exit: 72,704 bytes in 1 blocks
==1212==   total heap usage: 4 allocs, 3 frees, 74,728 bytes allocated
==1212==
==1212== LEAK SUMMARY:
==1212==    definitely lost: 0 bytes in 0 blocks
==1212==    indirectly lost: 0 bytes in 0 blocks
==1212==      possibly lost: 0 bytes in 0 blocks
==1212==    still reachable: 72,704 bytes in 1 blocks
==1212==         suppressed: 0 bytes in 0 blocks
==1212== Reachable blocks (those to which a pointer was found) are not shown.
==1212== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==1212==
==1212== For counts of detected and suppressed errors, rerun with: -v
==1212== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Virtual Destructors

To avoid this in general: **Any class with virtual member functions** should also have a `virtual` destructor, even if the destructor does nothing

Quiz - answers

Assume `class C` is derived from `class A` and `class B` and `class D` is derived from `class B`. If `class A` and `class B` both have `virtual` member functions, at the very least, the destructors of which classes must be `virtual`?

A. C and D

B. A and B

C. A, B and C

D. A, B, C and D

E. D only