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

#include "virt_dtor.h"

int main() {
    // Note use of base-class pointer
    Base *obj = new Derived();
    delete obj; // calls what destructor(s)?
    return 0;
}
```

`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

==9883== Memcheck, a memory error detector
==9883== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9883== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==9883== Command: ./virt_dtor
==9883==
==9883==
==9883== HEAP SUMMARY:
==9883==     in use at exit: 1,000 bytes in 1 blocks
==9883==   total heap usage: 4 allocs, 3 frees, 74,720 bytes allocated
==9883==
==9883== 1,000 bytes in 1 blocks are definitely lost in loss record 1 of 1
==9883==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-lin
==9883==    by 0x4007D6: Derived::Derived() (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermmedia
==9883==    by 0x400721: main (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermmediate Programming
==9883==
==9883== LEAK SUMMARY:
==9883==    definitely lost: 1,000 bytes in 1 blocks
==9883==    indirectly lost: 0 bytes in 0 blocks
==9883==      possibly lost: 0 bytes in 0 blocks
==9883==    still reachable: 0 bytes in 0 blocks
==9883==         suppressed: 0 bytes in 0 blocks
==9883==
==9883== For counts of detected and suppressed errors, rerun with: -v
==9883== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Virtual Destructors

```
// virt_dtor2.h

class Base {
public:
    Base() : base_memory(new char[1000]) { }
    virtual ~Base() { delete[] base_memory; }

private:
    char *base_memory;
};

class Derived : public Base {
public:
    Derived() : Base(), derived_memory(new char[1000]) { }
    virtual ~Derived() { delete[] derived_memory; }

private:
    char *derived_memory;
};
```

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

==9911== Memcheck, a memory error detector
==9911== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9911== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==9911== Command: ./virt_dtor2
==9911==
==9911==
==9911== HEAP SUMMARY:
==9911==     in use at exit: 0 bytes in 0 blocks
==9911==   total heap usage: 4 allocs, 4 frees, 74,728 bytes allocated
==9911==
==9911== All heap blocks were freed -- no leaks are possible
==9911==
==9911== For counts of detected and suppressed errors, rerun with: -v
==9911== 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