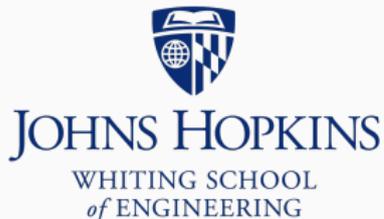


Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

Virtual destructors

```
class Base {  
public:  
    Base() : base_memory(new char[1000]) { }  
  
    ~Base() { delete[] base_memory; }  
  
private:  
    char *base_memory;  
};  
  
class Derived : public Base {  
public:  
    Derived() : Base(), derived_memory(new char[1000]) { }  
  
    ~Derived() { delete[] derived_memory; }  
  
private:  
    char *derived_memory;  
};
```

Virtual destructors

```
#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

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
$ valgrind ./virt_dtor
==22== Memcheck, a memory error detector
==22== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==22== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==22== Command: ./virt_dtor
==22==
==22== HEAP SUMMARY:
==22==    in use at exit: 1,000 bytes in 1 blocks
==22==   total heap usage: 4 allocs, 3 frees, 74,720 bytes allocated
==22==
==22== LEAK SUMMARY:
==22==    definitely lost: 1,000 bytes in 1 blocks
==22==    indirectly lost: 0 bytes in 0 blocks
==22==    possibly lost: 0 bytes in 0 blocks
==22==    still reachable: 0 bytes in 0 blocks
==22==         suppressed: 0 bytes in 0 blocks
==22== Rerun with --leak-check=full to see details of leaked memory
==22==
==22== For counts of detected and suppressed errors, rerun with: -v
==22== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

~Derived is *not* called; derived_memory is leaked

Virtual destructors

```
class Base {
public:
    Base() : base_memory(new char[1000]) { }

    // Now *** virtual ***
    virtual ~Base() { delete[] base_memory; }

private:
    char *base_memory;
};

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

    // Now *** virtual ***
    virtual ~Derived() { delete[] derived_memory; }

private:
    char *derived_memory;
};
```

Virtual destructors

```
#include "virt_dtor2.h"

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

Virtual destructors

```
$ g++ -o virt_dtor2 virt_dtor2.cpp
$ valgrind ./virt_dtor2
==28== Memcheck, a memory error detector
==28== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==28== Command: ./virt_dtor2
==28==
==28==
==28== HEAP SUMMARY:
==28==   in use at exit: 0 bytes in 0 blocks
==28==   total heap usage: 4 allocs, 4 frees, 74,728 bytes allocated
==28==
==28== All heap blocks were freed -- no leaks are possible
==28==
==28== For counts of detected and suppressed errors, rerun with: -v
==28== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Fixed; thanks to dynamic binding, delete obj calls ~Derived,
which in turn calls ~Base

Derived-class destructor always implicitly calls base-class destructor
at the end

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