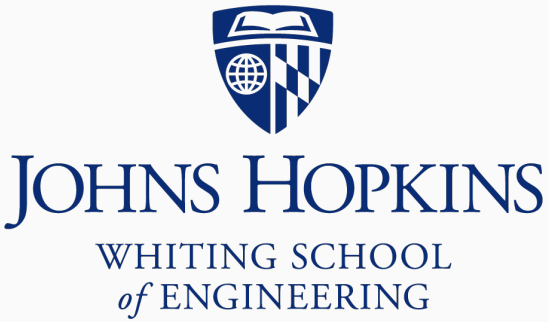
Destructors

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



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

Destructors

```
#include <cassert>

// What does this class do?  Anything wrong with it?
class Sequence {
public:
    Sequence(int sz) : array(new int[sz]), size(sz) {
        for(int i = 0; i < sz; i++) {
            array[i] = i;
        }
    }

    int at(int i) {
        assert(i < size);
        return array[i];
    }
private:
    int *array;
    int size;
};
```

Destructors

```
#include <iostream>
#include "sequence.h"

using std::cout;  using std::endl;

int main() {
    Sequence seq(10);
    for(int i = 0; i < 10; i++) {
        cout << seq.at(i) << ' ';
    }
    cout << endl;
    return 0;
}
```

```
$ g++ -c sequence_main.cpp -std=c++11 -pedantic -Wall -Wextra -g
$ g++ -o sequence_main sequence_main.o
$ ./sequence_main
0 1 2 3 4 5 6 7 8 9
```

Destructors

```
$ valgrind --leak-check=full ./sequence_main
0 1 2 3 4 5 6 7 8 9
==26== Memcheck, a memory error detector
==26== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==26== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==26== Command: ./sequence_main
==26==
==26==
==26== HEAP SUMMARY:
==26==     in use at exit: 40 bytes in 1 blocks
==26==   total heap usage: 3 allocs, 2 frees, 76,840 bytes allocated
==26==
==26== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==26==    at 0x4C308B7: operator new[](unsigned long) (vg_replace_malloc.c:423)
==26==    by 0x4009D4: Sequence::Sequence(int) (sequence.h:6)
==26==    by 0x4008FF: main (sequence_main.cpp:7)
==26==
==26== LEAK SUMMARY:
==26==    definitely lost: 40 bytes in 1 blocks
==26==    indirectly lost: 0 bytes in 0 blocks
==26==      possibly lost: 0 bytes in 0 blocks
==26==    still reachable: 0 bytes in 0 blocks
==26==         suppressed: 0 bytes in 0 blocks
==26==
==26== For counts of detected and suppressed errors, rerun with: -v
==26== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Destructors

Allocates new `int[sz]` in constructor, but never `delete[]`s it

It's common for a constructor to obtain a resource (allocate memory, open a file, etc) that should be released when object is destroyed

Destructor is a function called by C++ when the object goes out of scope or is otherwise deallocated (i.e. with `delete`)

Destructors

```
#include <cassert>

class Sequence {
public:
    Sequence(int sz) : array(new int[sz]), size(sz) {
        for(int i = 0; i < sz; i++) {
            array[i] = i;
        }
    }

    // *** Destructor (name starts with tilde) ****
    ~Sequence() { delete[] array; }

    int at(int i) {
        assert(i < size);
        return array[i];
    }
private:
    int *array;
    int size;
};
```

Destructors

```
$ g++ -c sequence_main.cpp -std=c++11 -pedantic -Wall -Wextra -g
$ g++ -o sequence_main sequence_main.o
$ ./sequence_main
0 1 2 3 4 5 6 7 8 9
```

Destructors

```
$ valgrind --leak-check=full ./sequence_main
0 1 2 3 4 5 6 7 8 9
==34== Memcheck, a memory error detector
==34== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==34== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==34== Command: ./sequence_main
==34==
==34==
==34== HEAP SUMMARY:
==34==     in use at exit: 0 bytes in 0 blocks
==34==   total heap usage: 3 allocs, 3 frees, 76,840 bytes allocated
==34==
==34== All heap blocks were freed -- no leaks are possible
==34==
==34== For counts of detected and suppressed errors, rerun with: -v
==34== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Destructors

Destructors are better than having a special member function for releasing resources; e.g.:

```
#include <cassert>

class Sequence {
public:
    ...
    // User must call clean_up when finished with Sequence
    void clean_up() { delete[] array; }
    ...
};
```

Destructors

Here, user forgets to call clean_up:

```
{
    Sequence s(40);
    // ... (no call to s.clean_up())
} // s goes out of scope and memory is leaked
```

More subtly:

```
{
    Sequence s(40);
    if(some_condition) {
        return 0; // memory leaked!
    }
    s.clean_up();
}
```