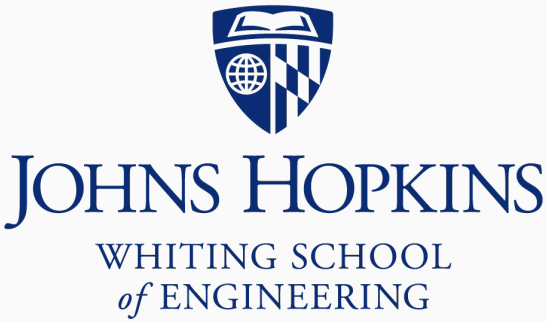
Lifetime & scope

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



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

Lifetime & scope

Variable's *lifetime* is the period when it exists in memory

- When lifetime ends, memory is reclaimed and can be reused for other variables

Variable's *scope* is the part of the program where you can use it

- Lifetime and scope are *often* the same *but not always*

Lifetime & scope

Scope and lifetime of a variable declared in a block {...} ends at terminal brace }

```
if(a == 7) {
    int c = 70;
    printf("%d\n", c);
}
```

When program reaches }, c is both *out of scope* (we can't refer to it anymore) and *dead* (memory reclaimed)

Lifetime & scope

For `for` loop index variable, scope is the loop body

```
for(int i = 0; i < 10; i++) {
    sum += i;
}
// after }, i is "dead" and "out of scope"
```

Lifetime & scope

Variables in scope at the time of a function call are *not* in scope in the callee

```c
#include <stdio.h>

void print_a() {
    // COMPILER ERROR; can't refer to 'a' here
    printf("a=%d\n", a);
}

int main() {
    int a = 1; // 'a' declared here
    print_a();
    return 0;
}
```

Lifetime & scope

Pointers give a way around this:

```
#include <stdio.h>

void print_a(int *a) {
    printf("*a=%d\n", *a); // OK
}

int main() {
    int a = 1; // a declared here
    print_a(&a);
    return 0;
}
```

```
$ gcc -o scope2 scope2.c -std=c99 -pedantic -Wall -Wextra
$ ./scope2
*a=1
```