Variables Scope and Lifetime

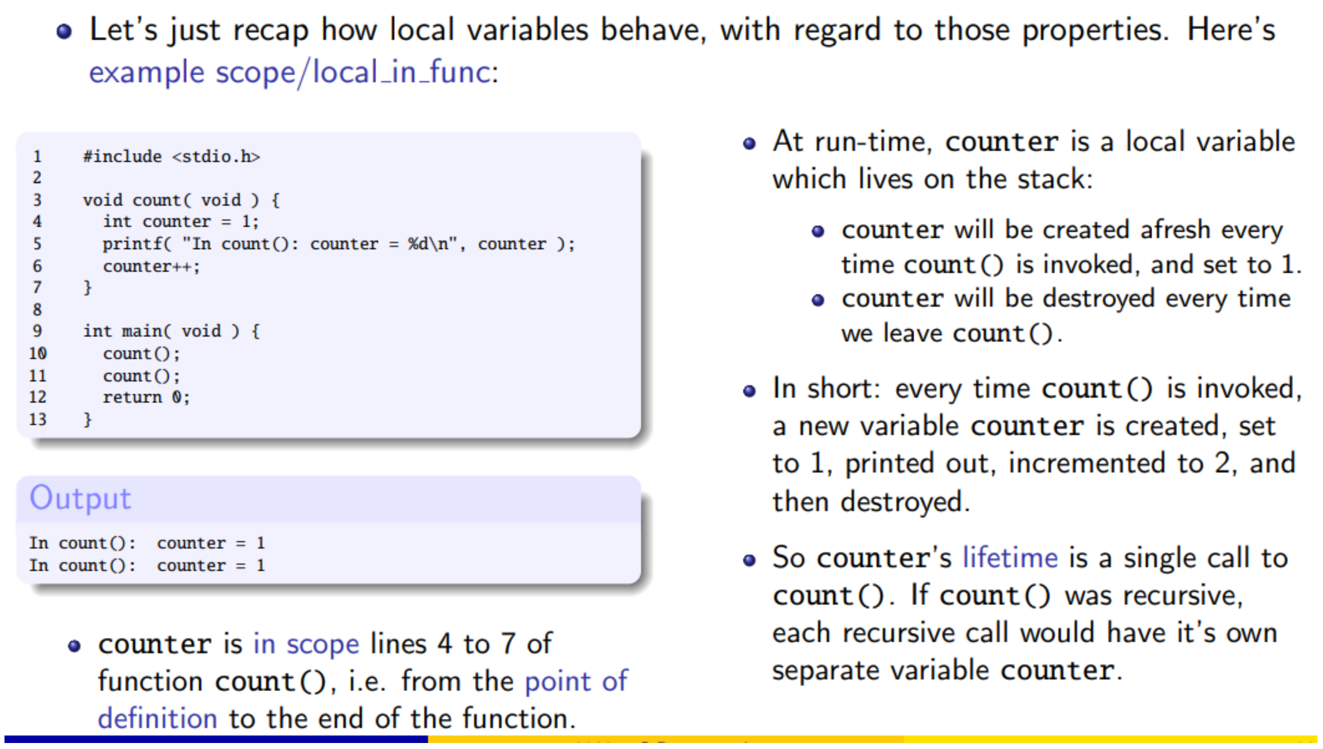
So far, we've only seen local variables. (Function parameters are just another kind of local variable.)

But C supports several other kinds of variables

There are two separate ways in which we can think about a variable to better understand it's behaviour:

- At compile time: which textual region of our source code our variable is visible in - the variable is said to be in scope in this region, described as a range of source lines
- At run time: what portion of the program runtime our variable is available, ie it's lifetime

Scope and lifetime are two very different views of a variable, and it's important to distinguish properly between them



- Let's just recap how local variables behave, with regard to those properties. Here's example scope/local_in_func:

```
1    #include <stdio.h>
2
3    void count( void ) {
4      int counter = 1;
5      printf( "In count(): counter = %d\n", counter );
6      counter++;
7    }
8
9    int main( void ) {
10     count();
11     count();
12     return 0;
13   }
```

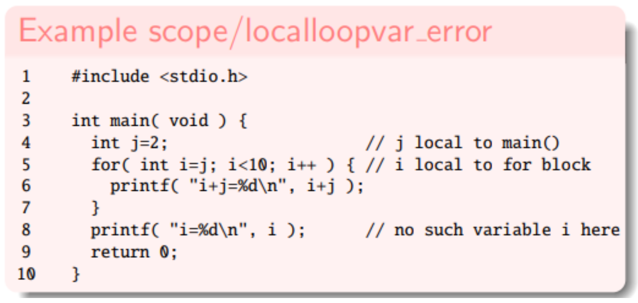
Output

```
In count():  counter = 1
In count():  counter = 1
```

- counter is in scope lines 4 to 7 of function count(), i.e. from the point of definition to the end of the function.
- At run-time, counter is a local variable which lives on the stack:
  - counter will be created afresh every time count() is invoked, and set to 1.
  - counter will be destroyed every time we leave count().
- In short: every time count() is invoked, a new variable counter is created, set to 1, printed out, incremented to 2, and then destroyed.
- So counter's lifetime is a single call to count(). If count() was recursive, each recursive call would have it's own separate variable counter.

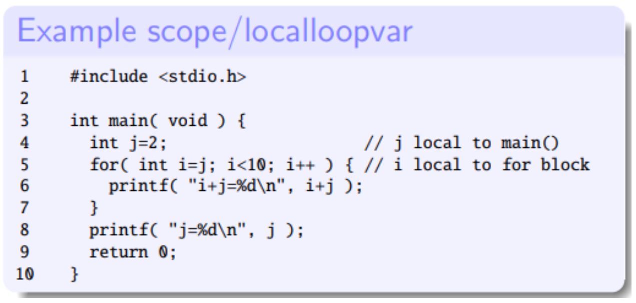
Nested Local scope

Note that local variables can have an even smaller scope than the current function - a for loop can declare it's own loop variable with an ultra-local scope:



Example scope/localloopvar_error

```
1   #include <stdio.h>
2
3   int main( void ) {
4     int j=2;                       // j local to main()
5     for( int i=j; i<10; i++ ) { // i local to for block
6       printf( "i+j=%d\n", i+j );
7     }
8     printf( "i=%d\n", i );        // no such variable i here
9     return 0;
10  }
```



Example scope/localloopvar

```
1   #include <stdio.h>
2
3   int main( void ) {
4     int j=2;                       // j local to main()
5     for( int i=j; i<10; i++ ) { // i local to for block
6       printf( "i+j=%d\n", i+j );
7     }
8     printf( "j=%d\n", j );
9     return 0;
10  }
```

In the left example, `j` is in scope between lines 4 and 10, and `i` is in scope between lines 5 and 7. Hence when we access `i` at line 8, our nested scope `i` is no longer in scope. As no other variable i is in scope, an error is generated

Global variables

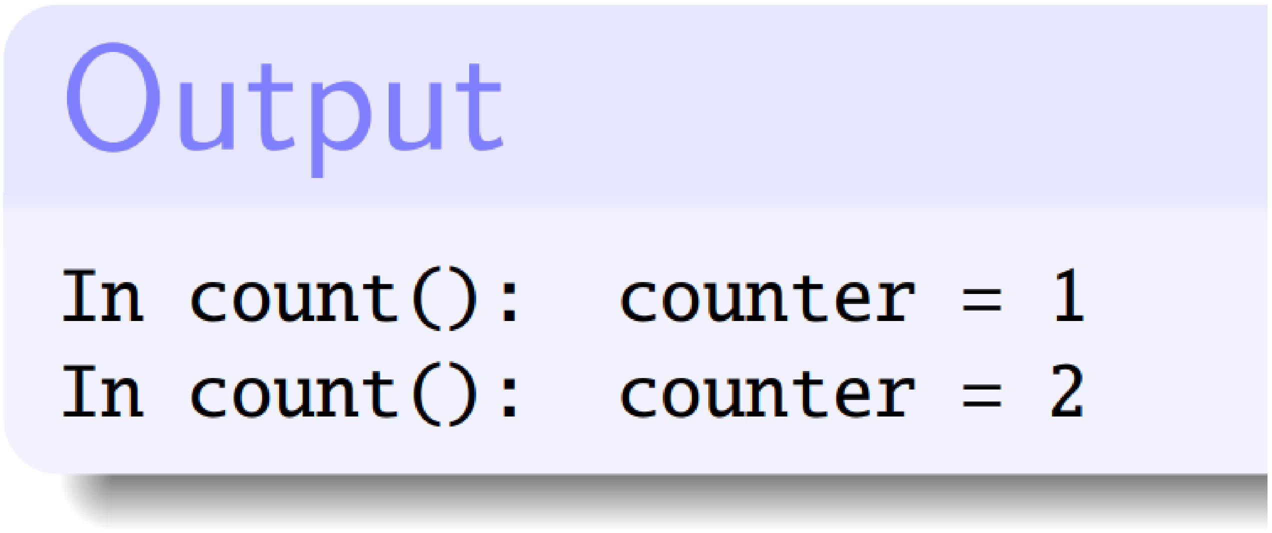
Any variable definition that is not inside a function is global:

```
#include <stdio.h>

int counter = 1;

void count( void ) {
        printf( "In count(): counter = %d\n", counter );
        counter++;
}

int main( void ) {
        count();
        count();
        return 0;
}
```



Output

```
In count():  counter = 1
In count():  counter = 2
```

It’s not only `count()` that can access our global `counter`: every function can. It’s scope is global - from line 3 to the end of the source file (line 14), and it’s lifetime is global (the entire runtime of the program) too

Global variables are harder to reason about than locals are, because any function may modify a global

But IMO globals are a valuable tool if not overused - especially to store some single main data structure without which none of the program code makes any sense

Give each global variable a comment at the point of definition saying exactly what purpose it serves

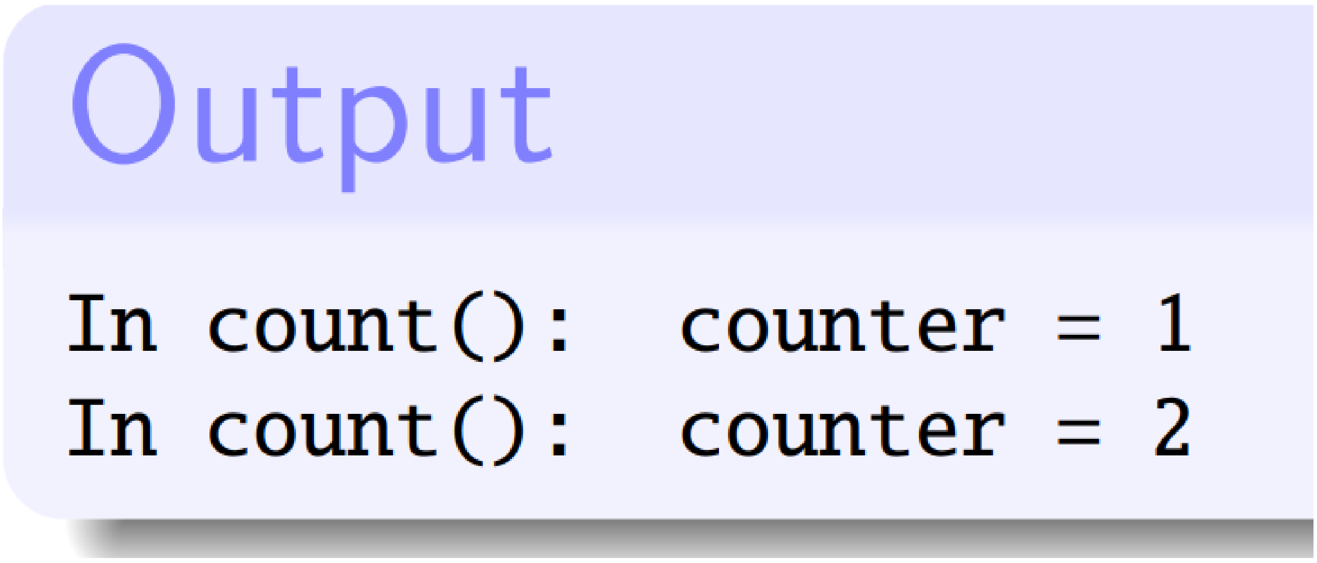
Static “Local” Variables - Local scope, Global lifetime

Suppose we want to have a variable that is “local in scope” to one function, but it’s value to be preserved across calls to that function - it’s lifetime should be global. As shown here in this example, we define counter inside a function but marked static:

```
#include <stdio.h>

void count( void ) {
  static int counter = 1;
  printf( "In count(): counter = %d\n", counter );
  counter++;
}

int main( void ) {
  count();
  count();
  return 0;
}
```



```
In count():  counter = 1
In count():  counter = 2
```

It’s scope is local (lines 4 to 7), but it’s lifetime is global - it will not be destroyed until the program ends. Effectively it becomes a new global variable with a hidden unique name, only accessible within `count()`