Automatic vs Static Variables

An introduction to automatic and static variables

We talked about variable **scope** and the idea that variables declared within a function are **local** to that function. What actually happens is that each time a function is called by another piece of code, all the variables declared within the function are created (that is, memory is allocated to hold them). When a function is finished, all of that local memory storage is de-allocated, and those variables essentially disappear. This is known as **automatic** local variables (they are automatically created and then destroyed as the function is called, and then finishes).

If you want local variables to persist, you can declare them as **static** local variables. You simply insert the word `static` in front of the variable type when you declare it inside your function. When declared in this way, the variable will **not be destroyed** when the function exits, but it (and its value) will persist. Next time the function is called, the value will have retained the value from the previous function call. It's a sort of global variable, but one that is still only accessible within the function in which it's declared.

Here's an example program that maintains a running count of the number of times the function `myFun()` has been called.

```
#include <stdio.h>

void myFunc(void) {
    static int num = 0;
    num++;
    printf("myFunc() has been called %d times so far\n", num);
}

int main() {
    myFunc();
    myFunc();
    myFunc();
//  printf("num = %d\n", num);  // THIS WOULD NOT WORK
    return 0;
}
```

When would you want to use **static** variables? One general case, like above, is when you want to keep track of the number of times a function has been called.

Another reason has to do with efficiency… if for example your function declares a **large** local variable whose values don't change from one function call to the next, it may be more efficient to declare it as static, so that it is created and initialized only once.

We've delved really deep into the workings of a function. I'll ask you an interesting question. Can we make a function where the number of arguments can vary?

The answer is yes. We'll look at these **variadic functions** now.