Applied C Programming

Prof. Georg Feil

Static local variables, Location of variables in memory

Summer 2018

Static local variables

The two meanings of the ‘static’ keyword

- So far we’ve seen that you can add the static keyword to global variable declarations, and function declarations
  - Makes that item visible in that module (.c file) only
  - Similar to ‘private’ in Java

- static can also be used with local variables in C
- A static local variable *remembers its value between function calls*
  - It is still only visible (usable) inside the function
  - This is a different meaning of static than we’ve seen

Two meanings of ‘static’ example

```c
#include <stdio.h>
#include <stdbool.h>

static int numberOfLoops = 10;    // How many times to execute the main loop
static void one(int num);

int main(int argc, char** argv)
{
    for (int i = 0; i < numberOfLoops; i++) {
        one(i);
    }
    return 0;
}

static void one(int num)
{
    static bool firstTime = true;    // Flag indicates first call to function

    if (firstTime) {
        printf("This is the first time calling one()\n");
        firstTime = false;
    }

    printf("Function one() called with parameter %d\n", num);
}
```

Visible only in this module

Visible only in this module

Remembers value between calls

Where Variables are Stored in Memory

Where variables are stored in memory

- The C compiler can put variables on the stack or in a static data area of memory, depending on
  - Where the variable is declared (inside or outside functions)
  - ‘static’ keyword
- Variables on the stack disappear when the function returns, others “stick around”

“Module” variable

Kind of variable	Storage Location
Global variable	Static data area
Global static variable	Static data area
Local variable	Stack
Static local variable	Static data area

Where variables are stored in memory

- It is sometimes useful to know and control where a variable is located in memory
  - In embedded systems memory is often limited (esp. stack space), so managing memory properly is important
- Simple, temporary variables inside functions should go on the stack (local variable, non-static)
  - Don’t make a variable non-local if it can be local
- Variables which need a large amount of memory (for example large arrays) should NOT go on the stack
- Variables whose value needs to be remembered across function calls should go in the static data area
  - Restrict the scope as much as possible – avoid global variables

Exercise 1

- Compile and run the following C program

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5000

int main(int argc, char** argv)
{
    double numbers[SIZE];   // Array of random numbers from 0 to 1
    for (int index = 0; index < SIZE; index++) {
        numbers[index] = (double) rand() / RAND_MAX;
    }

    double ave = 0;
    for (int index = 0; index < SIZE; index++) {
        ave += numbers[index];
    }
    printf("The average random number is %f\n", ave/SIZE);
}
```

- What happens if you make the array bigger… 50,000? 500,000?
- Try making the array 'numbers' static. Explain what's happening.