Tip 20: get to know static, automatic, manual memory

Ben Klemens

10 November 2011

level: intermediate
purpose: articulate why you hate C

C provides three models of memory management, which is two more than most languages and two more than you really want to care about.

static data is initialized before `main` starts. Array size is fixed at startup, but values can change (so it's not really static).

automatic is where you declare a varaible on first use, and it is removed when it goes out of scope. Most languages have only automatic-type data.

manual involves `malloc` and `free`, and is where most of your segfaults happen. This memory model is why Jesus weeps when he has to code in C.

Here's a little table of the differences in the three places you could put data:

	static	auto	manual
initialized on startup	•		
can be scope-limited	•	•	
set values on init	•	•	
`sizeof` measures array size	•	•	
persists across fn calls	•		•
can be global	•		•
set size at runtime		•	•
can be resized			•
Jesus weeps			•

Some of these things are features that you're looking for in a variable, like resizing or convenient initialization. Some of these things, like whether you get to set values on initalization, are technical consequences of the memory system. So if you want a different feature, like being able to resize in real time, suddenly you have to care about `malloc` and the pointer heap.

If we could bomb it all out and start over, we wouldn't tie together three sets of features with three sets of technical annoyances. But here we are.

All of this is about where you put your data. Variables are another level of fun:

1. If you declared your `char`, `int`, or `double` variable either outside of a function or inside a function with the `static` keyword, then it's static; otherwise it's automatic.

2. If you declared a pointer, the pointer itself has a memory type as per rule number one. But the pointer could be pointing to any of the three types of data. Static pointer to `malloc`ed data, automatic pointer to static data—all the combinations are possible.

Rule number two means that you can't identify the memory model by the notation. On the one hand, it's nice that we don't have to deal with one notation for auto arrays and one notation for manual arrays; on the other hand, you still often have to be aware of which you have on hand, so you don't get tripped up resizing an automatic array or not freeing a manual array. This is why the statement *C pointers and arrays are identical* is about as reliable as the rule about *i before e except after c*.

To do:
Check back on some code you have and go through the typology: what data is static memory, auto, manual; what variables are auto pointers to manual memory, auto pointers to static values, et cetera. If you don't have anything immediately on hand, try it with the tip on string literals (Entry #061).