Tip 51: Construct and destruct your structs

Ben Klemens

11 January 2012

level: writing object-centric code
purpose: Hide your `malloc`

If you were in a language that has abundant extra syntax for object-oriented programming, you might be absolutely required to write a *constructor* and *destructor* function for every object. There'd be a special syntax for it.

C doesn't have all that, so it's up to your own discipline to make this happen. It's still a good idea to have constructors and destructors, even if you don't use the pedantic names.

The reader will note that my advice here conflicts with the joyous use of throwaway structs in the last several tips. There are structs that you use to conveniently organize code as it goes in and out of a function or to provide a little more structure to a hundred-line segment of code, and there are structs that you use as the key players in a library or program.

A key difference in these throwaways and the structs upon which we base our work is that the more important structs will live far beyond the scope in which they were born. We are potentially passing the structure around to a great number of subfunctions, each of which may modify it, so the more efficient and less error-prone route is to set these things up as pointers. And so, it has come time to invoke the name of the beast: ¡`malloc()`!

Having new/copy/free functions mean that your memory management worries are pretty brief: in the new and copy functions, allocate the memory with `malloc()`; in the free function, remove the structure with `free()`, and having set up these functions, do these things only with the given new/copy/free functions.

Next time, some nuance about making this work and not double-freeing, and examples.

To do:
Now that you've seen all the advice to date about how you can allocate arrays and structures without `malloc` and you have only a handful of `malloc`s and `free`s left in your code, here's an experiment you can try: wrap 100% of your calls to these two functions in new/copy/free functions.