Tip 48: limit typedef scope

Ben Klemens

5 January 2012

level: your functions take functions as arguments
purpose: keep the code clean through that kind of mess

Next time we'll start writing throwaway structs, which are a pretty common form in C. Elements that are global to everything add cognitive load and can be hard to work with, so keep that stuff limited, but over the course of one screen of text have a ball, and generate whatever scaffolding you deem to make the situation more readable.

Or, if you've read a lot of academic papers, then you know how they are structured: before getting serious work done, there's a definitions section. Some papers explicitly have *Definitions* as a header; some just start slow by defining every term at first and then building up to the real work. Code that adds local definitions of immediate relevance is easily more readable and more maintainable than code that doesn't clutter up the namespace but at the same time doesn't explain itself.

But before we get to writing our throwaway structs to define local structures, here's one little point about typedefs that makes these throwaways possible.

The scope of a `typedef` is the same as the scope of any other declaration. That means that you can `typedef` things inside a single file and not worry about them cluttering up the namespace, and you might even find reason to have `typedef`s inside of a single function.

Conversely, if you would like a `typedef` to have broader scope, then add it to a header file, which you would `#include` as needed.

Here's some sample code designed to fail.

[Notice how I used a backslash to split `<<'---'` into two lines. I think this looks spiffy, and it's what I do in all my shell scripts, but make sure when cutting and pasting that there's nothing after the backslash.]

```
# This will fail:
cat > f1.c << \
'---------'
typedef struct {
    int a1, a2;
} astruct;
---------

cat > f2.c << \
'---------'
```

```
astruct a; //type declared in the othe file.

int donothing(){
    typedef struct {int d1, d2;} dstruct;
}

dstruct d; //type declared in the above function.

int main(){};
---------

gcc f1.c f2.c
```

OK, that’s all. Over the next few episodes, we’ll use that.