Tip 71: Don't confuse pointer declarations with pointer uses

Ben Klemens

21 February 2012

level: confused by pointers
purpose: be less confused by pointers

The ostensible rationale for the pointer declaration syntax is that the use and the declaration look alike. What they mean by this is that when you declare

```
int *i;
```

then `*i` is an integer, so it's only natural that we'd declare that `*i` is an integer via `int *i`.

So that's all well and good, and if it helps you, then great. I'm not sure if I could invent a less ambiguous way of doing it.

Here's a common design rule, espoused throughout The Design of Everyday Things[1], for example: *if things have different function, then make them look different.* That book gives the example of airplane controls, where two identical levers often do drastically different things.

Here, C syntax crashes and burns, because `*i` in a declaration and `*i` outside of a declaration do very different things. Examples:

```
int *i = 23;         //wrong
*i = 23;             //right
int *i = malloc(1); //right
```

I've thrown the rule that declaration looks like usage out of my brain. Here's the rule I use, which has served me well: *in a declaration, a star indicates a pointer; off declaration, a star indicates the value of the pointer.*

Here is a valid snippet:

```
int i = 13;
int *j = &i;
int *k = j;
*j = 12;
```

[1] `http://www.jnd.org/books.html#33`

Using the rule above, you can see that on line two, the initialization is correct because `*j` is a declaration, and so the address of a pointer. On line three, `*k` is also the address of a declaration, so it makes sense to assign it to `j`, the address of a pointer. On the last line, `*j` is not in a declaration, so it indicates a plain integer, and so we can assign 12 to it (and `i` will change as a result).

OK, that's all today: bear in mind that when you see `*i` on a declaration line, it is a pointer-to-something; when you see `*i` on a non-declaration line, it is the pointed-to value.