Tip 40: Designated initializers

Ben Klemens

20 December 2011

level: `struct` user
purpose: understand your own code

I'm going to define by example today. Here is a short program that just prints a box to the screen. You get to specify whether you want the star to be in the upper right, left, center, et cetera by setting up a `direction_t` structure. [I made `up` and `down` `double`s just for the sake of having multiple types in the struct.]

The printing is simple, and is cleaned up by what I called the Obfuscatory If in *Modeling with Data*. [I was too harsh.]

The focus is in `main`, where we declare three of these structures using designated initializers—i.e., we designate the name of each structure element in the initializer. I'll meet you on the other end of the code to discuss further.

```
#include <stdio.h>

typedef struct {
    int left, right;
    double up, down;
} direction_t;

void this_row(direction_t d); //see below
void draw_box(direction_t d);

int main(){
    printf("left:");
    direction_t D = {.left=1};
    draw_box(D);

    printf("upper right:");
    D = (direction_t) {.up=1, .right=1};
    draw_box(D);

    printf("center:");
    draw_box((direction_t){});
```

```
}

/* Print a box. Don't bother reading these
    if you don't feel like it. */
void this_row(direction_t d){
    printf( d.left    ? "*..\n"
            : d.right ? "..*\n"
            : ".*.\n");
}

void draw_box(direction_t d){
    printf("\n");
    d.up                ? this_row(d) : printf("...\n");
    (!d.up && !d.down)  ? this_row(d) : printf("...\n");
    d.down              ? this_row(d) : printf("...\n");
    printf("\n");
}
```

The old school alternative was to memorize the order of struct elements and initialize all of them without any labels, so the upper right declaration would be:

```
direction_t upright = {0, 1, 1, 0};
```

This is illegible and makes people hate C. Outsize of the rare situation where the order is natural and obvious, please consider the unlabelled form to be deprecated.

Notice also that in the setup of the upper right struct, I had designated elements out of order relative to the proper structure. Life is too short to remember the order of arbitrarily-ordered sets—let the compiler sort 'em out.

What about the elements not declared? They're initialized to zero. No elements are left undefined.

Remember compound literals (Entry #074), which I'd introduced in terms of arrays? Being that structs are just arrays with named and oddly-sized elements, you can use them for structs too, as I did in the *upper right* and *center* structs in the sample code. As before, you need to add a cast-like `(typename)` before the curly braces. The first example in `main` is a direct declaration, and so doesn't need the compound initializer syntax.

To do:
Rewrite every struct declaration in all of your code to use designated initializers. I mean this—the old school way was terrible. Notice also that you can rewrite junk like

```
direction_t D;
D.left = 1;
D.right = 0;
D.up = 1;
D.down = 0;
```

with

```
direction_t D = {.left=1, .up=1};
```