Tip 24: Compound literals

Ben Klemens

18 November 2011

level: medium
purpose: create fewer one-off temp variables; lots of future uses

I know, you have no idea what the title means, but thanks for clicking through anyway.

You can write a single element into your text easily enough—C has no problem understanding `f(34)`.

But if you want to send a list of elements as an argument to a function—a compound literal value like `{20.38, a_value, 9.8}`—then there's a syntactic caveat: you have to put a sort of type-cast before the compound literal, or else the parser will get confused. The list now looks like this: `(double[]) {20.38, a_value, 9.8}`, and the call looks like

```
f((double[]) {20.38, a_value, 9.8});
```

To give a full example, say that we have a function `sum` that takes in an array of `doubles`. Then here are two ways for `main` to call it:

```
#include <math.h> //NAN
#include <stdio.h>

double sum(double  in[]){
    double out=0;
    for (int i=0; !isnan(in[i]); i++) out += in[i];
    return out;
}

int main(){
    double *list = (double[]){1.1, 2.2, 3.3, NAN};
    printf("sum: %g\n", sum(list));

    printf("quick sum: %g\n", sum((double[]){1.1, 2.2, 3.3, NAN}));
}
```

The first two lines of `main` generate a single-use array named `list` and then send it to `sum`; the last line does away with the incidental variable and goes straight to using the list.

[This paragraph is arcana; you are welcome to skip it.] The first line in `main` is the typical example of an initialization via a compound literal. ¿How does it differ from

```
double alist[] = {0.1, 0.3, 0.5, NAN};
```

? This is back to the obscureness of the last tip (Entry #073). For the typical array intialization, we have an auto-allocated array and it is named `alist`. The compound initializer generates an anonymous auto-allocated array, and then we immediately point a pointer to it. So `alist` *is* the array, while `list` is a pointer to an anonymous array. May you never be in a position where you have to care about this distinction.

There's your intro to C.I.s; I'll demonstrate many uses over the coming weeks. Meanwhile, ¿does the code on your hard drive use any quick throwaway lists whose whose use could be streamlined by a compound initializer?