Tip 44: Brevity is the soul of incomprehensibility

Ben Klemens

28 December 2011

level: useful anywhere
purpose: don't let elegance give you bad style

Here's a rewrite of the tip from last time. As per Tip #7, you will need to have `-include apop.h` on the compiler's command line for this to work. It takes the dot product of a vector with a permutation matrix.

```
int main(){
    apop_data_show(
        apop_dot(
           &(apop_data){.matrix=&(gsl_matrix){.size1=3, .size2=3, .tda=3,
                               .data = (double[]){1, 0, 0,
                                                  0, 0, 1,
                                                  0, 1, 0
                               }}},
           &(apop_data){.vector=&(gsl_vector){.size=3, .stride=1,
                               .data = (double[]){1, 2, 3}}}
        )
    );
}
```

Finally, we've reached the culmination of so many tips about shaving a line or two off the program: ¡this is one line of code! There are no state variables and no declarations. Designated initializers and compound literals allow us to insert complex structures in place, without pausing ahead of time to set them up and give them a name. The `apop_dot` function allocates memory for a new `apop_data` set, but at this point I've made clear that (outside of code called thousands of times) freeing memory smaller than a kilobyte is just a waste of time, so I don't keep track of the structure `apop_dot` returns.

After all that work, we got our wish: our line count is as small as the natural numbers get. We can add C one-liners to the wealth of Perl, Python, Mathematica, awk (and so on) one-liners which your search engine will present to you in abundance.

[In textbook LISP-like languages, functions are *typically* one liners. *The Structure and Interpretation of Computer Programs* gets a full two hundred pages through before needing a state variable. My impression is that real-world LISP-like code is rarely written like that.]

As well as achieving full brevity and sparing us the annoyance of declaring variables, this program is unreadable. It is unreadable not just because of the junk about having to give the stride and TDA for the GSL's structures of the slightly too verbose compound literal syntax, but for the very reason that nothing has a name.

A good name provides documentation as to what a thing does. Try describing the above code to your imaginary friend, and you will find yourself using sentences, with nouns and verbs. If an element plays a relevant part in your description, it probaly merits having a name.

Here's another rewrite. It's still hard to read how the GSL's data structures are formed, but I wrote last time about how that'll always be the case—at least that stuff is segregated from everything else so you can make a clear decision about replacing it (which you should).

```
int main(){
    double transformation_matrix[] = {1, 0, 0,
                                      0, 0, 1,
                                      0, 1, 0};
    double vector[] = {1, 2, 3};

    //I'll keep the GSL step, though apop_data_fill works.
    gsl_matrix m = { .data = transformation_matrix,
                        .size1=3, .size2=3, .tda=3};
    gsl_vector v = { .data = vector, .size=3, .stride=1};

    //I decided that the GSL-as-apop elements didn't merit a
    //name. This is of course subjective.
    apop_data *out = apop_dot(
           &(apop_data){.matrix=&m},
           &(apop_data){.vector=&v}
        );

    apop_data_show(out);
}
```