

Tip 56: Enums—don't bother

Ben Klemens

21 January 2012

level: You have laundry lists

purpose: don't force users to memorize your laundry lists

Enums are a good idea that went bad.

The benefit is clear enough: integers are not at all mnemonic, and so wherever you are about to put a short list of integers in your code, you are better off naming them. Here's (the even worse means of) how we could do it without the `enum` keyword:

```
#define NORTH 0
#define SOUTH 1
#define EAST 2
#define WEST 3
```

With `enum`, we can shrink that down to one line of source code, and our debugger is more likely to know what `EAST` means; here's the improvement over the sequence of `#defines`:

```
enum directions {NORTH, SOUTH, EAST, WEST};
```

But we now have five (5) new symbols in our global space, including `directions` and `NORTH`, `SOUTH`, et al.

For an `enum` to be useful, it typically has to have global scope. For example, you'll often find `enums` typedefed in the public header file for a library. Having a global variable creates responsibilities. To minimize the chance of namespace clashes, library authors use names like `G_CONVERT_ERROR_NOT_ABSOLUTE_PATH` or the relatively brief `CblasConjTrans`. I have to look them up every time and they take up half the line.

At which point an innocuous and sensible idea has fallen apart. I don't want to type these messes, and I use them so infrequently that I have to look them up every time (especially since many are infrequently used error values or input flags). Also, all caps continues to read like yelling.

My own habit is to use single characters, wherein I would mark transposition with `'t'` and a path error with `'p'`. I think this is enough to be mnemonic—in fact, I'm far more likely to remember how to spell `'p'` than how to spell the above all-caps mess—and it requires no new entries in the namespace.

Before you start arguing easy-to-parody efficiency issues, bear in mind that an enumeration is typically an integer, while `char` is really just C-speak for a single byte. So when comparing enums, you will likely need to compare the states of about sixteen bits, while with a `char`, you need compare only eight. ‘A two-fold speed gain!, dropping the time for this comparison from effectively zero and not worth tracking to effectively zero and still not worth tracking.

We sometimes need to combine flags. When opening a file using the `open` system call, you may need to send `O_RDWR|O_CREAT`, which is the bitwise combination of the two enums. You probably don’t use `open` directly all that often; you are probably making more use of `fopen`, which is more user friendly. Instead of using an enum, it uses a one- or two-letter string, like `"r"` or `"r+"` to indicate whether something is readable, writeable, both, et cetera.

In the context, you know `"r"` stands for *read*, and if you don’t have the convention memorized, you can confidently expect that you will after a half-dozen more uses of `fopen`. Whereas I still have to check whether I need `CblasTrans`, `CBLASTrans` `CblasTranspose`, ..., every time.

Again, caring about the runtime efficiency thing is ‘70s. The twenty seconds it takes to look up an awkward enum times a dozen re-lookups is equivalent to a few billion `strcmps` between two two- or three-letter strings. If you really think it matters, then, as above, you’d rather use a single character than an enum.

There are reasons for using enums: sometimes you have an array that makes no sense as a struct but that nonetheless requires named elements; when doing kernel-level work, giving names to bit patterns is essential. But in those cases where enums are used to indicate a short list of options or a short list of error codes, a single character or a short string can serve the purpose without cluttering up the namespace or authors’ memory.