Tip 25: Variadic macros

Ben Klemens

20 November 2011

level: not hard
purpose: delightful tricks to follow in the coming week

I broadly consider variable-length functions in C to be broken—I have a personal rules about the three forms I'm willing to use, which I'll maybe expound upon later.

But variable-length macro arguments are easy. The keyword is `__VA_ARGS__`, and it expands to whatever set of elements were given.

For example, here's a fast way to implement a customized variant of printf for reporting errors:

```
#define print_a_warning(...) {\
    printf("Glitch detected in %s at line %s:%i: ", __FUNCTION__, __FILE__, __LI
    printf(__VA_ARGS__); }

//usage:
print_a_warning("x has value %g, but it should be between zero and one.\n", x);
```

The `__FUNCTION__`, `__FILE__`, and `__LINE__` macros get filled in with what you'd expect.

You can probably guess how the ellipsis (`...`) and `__VA_ARGS__` work: whatever is between the parens gets plugged in at the `__VA_ARGS__` mark.

You can have arguments before the ellipsis if you want. A full example:

```
#define print_a_warning(dostop, ...) { \
    printf("Glitch detected in %s at line %s:%i: ", __FUNCTION__, __FILE__, __LI
    printf(__VA_ARGS__);    \
    if (dostop=='s') abort(); \
}

int main(int argc, char ** argv){
    if (argc <= 1) print_a_warning('s', "argc has value %i, but it should be two
    if (argc > 2) print_a_warning('c', "argc has value %i, but it should be two.

    printf("arg one = %s\n", argv[1]);
}
```