Tip 27: Foreach in C

Ben Klemens

24 November 2011

level: medium
purpose: borrow a useful scripting language construct

Last time, you saw that you can use a *compound literal* anywhere you would put an array or structure.

For example, here is an array of strings declared via a compound literal:

```
char **strings = (char*[]){"Yarn", "twine"};
```

The compound literal is automatically allocated, meaning that you need neither `malloc` nor `free` to bother with it. At the end of your function it just disappears.

Now let's put that in a `for` loop. The first element of the loop declares the array of strings, so we can use the line above. Then, we step through until we get to the `NULL` marker at the end. For additional comprehensibility, I'll `typedef` a string type, as per Tip 17 (Entry #067).

```
#include <stdio.h>

typedef char* string;

int main(){
    string str = "thread";
    for (string *list = (string[]){"yarn", str, "rope", NULL}; *list; list++)
        printf("%s\n", *list);
}
```

It's still noisy, so let's hide all the syntactic noise in a macro. Then `main` is as clean as can be:

```
#include <stdio.h>
//I'll do it without the typedef this time.

#define foreach_string(iterator, ...)\
    for (char **iterator = (char*[]){__VA_ARGS__, NULL}; *iterator; iterator++)

int main(){
```

```
    char *str = "thread";
    foreach_string(i, "yarn", str, "rope"){
        printf("%s\n", *i);
    }
}
```

To do:
Rewrite the macro to use `void` pointers rather than strings. If you're the sort of person who never tries even the easy exercises when reading tutorials then (1) I hate you, and (2) just wait until next time, when I'll use the solution to this as part of the next tip.