Tip 84: Use m4 to automate OOP boilerplate

Ben Klemens

18 March 2012

level: macro hacker, bored with OOP
purpose: get to the good stuff
prerequisite: Last episode, where we used m4 to generate HTML (Entry #133)

Every object needs a `typedef`, a new/copy/free function, and a header for public use. These functions tend to be pretty boilerplate, and every language that expects its users to generate new objects all day long has some means of autogenerating this stuff. Some have an elaborate catalog of templates; some expect that every IDEs will have a means of generating all this crap via a text editor macro.

I wrote some C preprocessor macros to do all this at one point, but here's another approach, via m4, a little macro language which does nothing better than generate boilerplate.

Of course, you're still going to have to do some work, inventing the elements that go into the struct, and deciding how they get allocated, copied, and freed, but you have a framework to put that in for free. You might not like my boilerplate object declarations anyway. As with many of these blog entries, this is more to give you a potentially useful concept and enough code to start hacking it to work the way you do.

OK, enough with the caveats. Here's the m4 code for you. It's self-documenting, especially if you read last episode, where we used m4 to generate HTML (Entry #133).

```
m4_divert(-1)

    MekeHeader(objname) generates a header file with a
    typedef and new/copy/free elements.

    MakeObject(objname) generates the boilerplate new/copy/free
    functions themselves.

    Usage:

    echo "MakeHeader(objname)" | m4 -P objectify.m4 '-'  > obj.h
    echo "MakeObject(objname)" | m4 -P objectify.m4 '-'  > obj.c

    There are markers for where the actual content goes.
```

```
m4_changequote(‘’,‘’)
m4_changecom(m4 comment:)

m4_define(MakeHeader, m4_dnl
typedef struct $1 {
   //Place elements here.
} $1;

$1 *$1_new(void);
$1 *$1_copy($1 *in);
void $1_free($1 *in);)

m4_define(MakeObject,#include "$1.h"
$1 *$1_new( ){
    $1 *out = malloc(sizeof($1));
    *out = ($1){ };
    return out;
}

$1 *$1_copy($1 *in){
    $1 *out = malloc(sizeof($1));
    *out = *in;
    //Element pointers and other copying happens here
    return out;
}

void $1_free($1 *in){
    //free subelements here
    free(in);
})

m4_divert(0)m4_dnl
```