Tip 10: Use `asprintf` to make string handling less painful

Ben Klemens

21 October 2011

level: basic string user
purpose: don't call `malloc`

The function allocates the amount of string space you will need, and then fills the string. That means you never really have to worry about string-allocing again.

`asprintf` is not part of the C standard, but it's available on systems with the GNU or BSD standard library, which covers pretty much everybody (including Apple computers).

The old way made people homicidal (or suicidal, depending on temprament) because you first have to get the length of the string you are about to fill, allocate space, and then actually write to the space. ¡Don't forget the extra slot in the string for the null terminator!

This sample program demonstrates the painful way of setting up a string, for the purpose of using C's `system` command to run an external program. The thematically appropriate program, `strings`, searches a binary for printable plain text. I'm assuming that you're compiling to `a.out`, in which case the program searches itself for strings. This is perhaps amusing, which is all we can ask of demo code.

```
#include <stdio.h>
void get_strings(char *in){
    char *cmd;
    int len = strlen("strings ") + strlen(in) + 1;
    cmd = malloc(len); //The C standard says sizeof(char)==1, b.t.w.
    snprintf(cmd, len, "strings %s", in);
    system(cmd);
}

int main(){
    get_strings("a.out");
}
```

[Apophenia users, use `apop_system("strings %s", in)` to bypass the need to first print to a string and then call `system`. Everybody else, feel free to write your own `system_with_printf` function; it's not that hard.]

With `asprintf`, `malloc` gets called for you, which means that you also don't need the step where you measure the length of the string:

```
#include <stdio.h>
void get_strings(char *in){
    char *cmd;
    asprintf(&cmd, "strings %s", in);
    system(cmd);
}

int main(){
    get_strings("a.out");
}
```

The actual call to `asprintf` looks a lot like the call to `sprintf`, except you need to send the location of the string, not the string itself, because something new will be `malloc`ed into that location. Also, we don't need to send the length parameter like with `snprintf`, because `asprintf` is smart enough to count letters for you.