Tip 11: String literals

Ben Klemens

23 October 2011

level: intermediate string user
purpose: understand an annoying subtlety of C string handling

Here is a program that sets up two strings and prints them to the screen:

```
#include <stdio.h>
int main(){
    char *s1 = "Thread";

    char *s2;
    asprintf(&s2, "Floss");

    printf("%s\n", s1);
    printf("%s\n", s2);
}
```

Both forms will leave a single word in the given string. However, the C compiler treats them in a very different manner, which can trip up the unaware.

Did you try the sample code in tip #10 that showed what strings are embedded into the program binary? In the example here, `Thread` would be such an embedded string, and `s1` could thus point to a location in the executable program itself. How efficient—you don't need to spend run time having the system count characters or waste memory repeating information already in the binary. I suppose in the 1970s this mattered.

Both the baked-in `s1` and the allocated-on-demand `s2` behave identically for reading purposes, but you can't modify or free `s1`. Here are some lines you could add to the above example, and their effects:

```
s2[0]='f'; //Switch Floss to lowercase.
s1[0]='t'; //Segfault.

free(s2); //Clean up.
free(s1); //Segfault.
```

If you think of a bare string declared like `"Floss"` as pointing to a location in the program itself, then it makes sense that `s1`'s contents will be absolutely read-only.

[I honestly don't know how your compiler really handles a constant string, but it is a fine mental model to presume it is pointing to a point in the program, so writing upon is strictly forbidden.]

Did you think this would be a series about why C is better than every other language in every way? If so, sorry to disappoint you. The difference between constant and variable strings is subtle and error-prone, and makes hard-coded strings useful only in limited contexts. I can't think of a scripting language where you would need to care about this distinction.

But here is one simple solution: `strdup`, which is POSIX-standard, and is short for *string duplicate*. Usage:

```
char *s3 = strdup("Thread");
```

The string `Thread` is still hard-coded into the program, but `s3` is a copy of that constant blob, and so can be freely modified as you wish.