A short asprintf implementation

Ben Klemens

17 Feb 2014

In 21st Century C[1] (and a few blog posts beginning with this one (entry #060)), I raved about `asprintf`, a variant of `sprintf` in the GNU standard library that will

- Count how many bytes your printf statement and its arguments would expand to once formatted.
- Allocate a string that long
- Fill that string.

I was so enthusiastic about a nonstandard function because it closes serious security loopholes while solving a common annoyance. It makes string-handling in C pleasant.

I got feedback from some readers that they did not want to paste the code for `asprintf` into their own projects. The source is provided by the GNU in its Libiberty package via the permissive LGPL license, but I am not going to argue with personal preference.

But the problem remains: counting the length that a `printf` statement and its arguments will eventually expand to is error-prone, so how can we get the computer to do it for us? The answer has been staring at us all along, in C99 §7.19.6.12(3) and C11 §7.21.6.12(3): "The `vsnprintf` function returns the number of characters that would have been written had n been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred."

So if we do a test run with `vsnprintf` on a one-byte string, we can get a return value with the length that the string should be. Then we can allocate the string to that length and run `vsnprintf` for real. We're running the function twice, so it may take twice as long to work, but it's worth it for the safety and convenience.

Ideally, I'd be writing around the safer `vsnprintf_s` function, but a lot of standard libraries don't yet support it (standard libraries tend to evolve more slowly than compilers, because there's more work to do).

OK, here is the implementation. Paste it into your code base, or if you've saved it as `a_sprintf.c` compile it as a test program with

```
CFLAGS='-g -Wall -std=c99 -DTest_asprintf' make a_sprintf
```

I wrapped it in a `HAVE_ASPRINTF` check to be Autoconf-friendly.

[1] `http://tinyurl.com/C-for-moderns`

```
#ifndef HAVE_ASPRINTF
#include <stdio.h> //vsnprintf
#include <stdlib.h> //malloc
#include <stdarg.h> //va_start et al

/* The declaration, to put into a .h file. The __attribute___ tells the compiler to check printf-
      style type-compliance.
   It's not C standard, but a lot of compilers supprt it. Just remove it if yours doesn't */

int asprintf(char **str, char* fmt, ...) __attribute__ ((format (printf,2,3)));


int asprintf(char **str, char* fmt, ...){
    va_list argp;
    va_start(argp, fmt);
    char one_char[1];
    int len = vsnprintf(one_char, 1, fmt, argp);
    if (len < 1){
        fprintf(stderr, "An encoding error occurred. Setting the input pointer to NULL.\n");
        *str = NULL;
        return len;
    }
    va_end(argp);

    *str = malloc(len+1);
    if (!str) {
        fprintf(stderr, "Couldn't allocate %i bytes.\n", len+1);
        return -1;
    }
    va_start(argp, fmt);
    vsnprintf(*str, len+1, fmt, argp);
    va_end(argp);
    return len;
}
#endif

#ifdef Test_asprintf
int main(){
    char *s;
    asprintf(&s, "hello, %s.", "Reader");
    printf("%s\n", s);

    asprintf(&s, "%c", '\0');
    printf("blank string: [%s]\n", s);

    int i = 0;
    asprintf(&s, "%i", i++);
    printf("Zero: %s\n", s);
}
#endif
```