Tip 12: Use asprintf to extend strings

Ben Klemens

25 October 2011

level: basic string user
purpose: `malloc` will be lonely, because you never call it

Here is an example of the basic form for appending another bit of text to a string using `asprintf`, which, as per tip #10, can be your workhorse for string handling:

```
asprintf(&q, "%s and another_clause %s", q, addme);
```

I (heart) this for generating queries. I would put together a chain something like this contrived example:

```
int row_number=3;
char *q =strdup("select ");
asprintf(&q, "%s row%i \n", q, row_number);
asprintf(&q, "%s from tab \n", q);
asprintf(&q, "%s  where row%i is not null", q, i);
```

And in the end I have

```
select row3
from tab
where row3 is not null
```

A rather nice way of putting together a long and painful string. [I had trouble coming up with a simple example for this one that didn't look contrived. But when each clause of the query requires a subfunction to write by itself, this sort of extend-the-query form starts to make a lot of sense. Apophenia users, see also `apop_text_paste`.]

But it's a memory leak, because the blob at the original address of `q` isn't released when `q` is given a new location by `asprintf`. For one-off string generation, it's not even worth caring about—you can drop a few million query-length strings on the floor before anything noticeable happens.

If you are in a situation where you might produce an unknown number of strings of unknown length, then you will need a form like this:

```
//Safe asprintf macro
#define Sasprintf(write_to,  ...) {\
    char *tmp_string_for_extend = write_to;    \
```

```
    asprintf(&(write_to), __VA_ARGS__);    \
    free(tmp_string_for_extend);  \
}

//sample usage:
int main(){
    int i=3;
    char *q = NULL;
    Sasprintf(q, "select * from tab");
    Sasprintf(q, "%s where row%i is not null", q, i);
    printf("%s\n", q);
}
```

Discussion and caveats:
The `Sasprintf` macro, plus occasional use of `strdup`, is enough for roughly 100% of your string-handling needs. Except for one glitch and the occasional `free`, you don't have to think about memory issues at all.

The glitch is that if you forget to initialize `q` to `NULL` or via `strdup` then the first use of the `Sasprintf` macro will be freeing whatever junk happened to be in the uninitialized location `q`—a segfault.

As you learned in the last tip, the following also fails—wrap that declaration in `strdup` to make it work:

```
char *q = "select * from";
Sasprintf(q, "%s %s where row%i is not null", q, tablename, i);
```