Tip 70: Parse text with `strtok`

Ben Klemens

19 February 2012

level: text handler
purpose: Do the easy parsing easily

Tokenizing is the simplest and most common parsing problem, in which we split a string into parts at delimiters. If the delimiter is whitespace like `" \t\n\r"` then you're splitting words; or you might have a path, like `/usr/include:/usr/local/include:."` to split at the colons; or a simple newline, `\n`, will split text into lines; or your configuration file may have the form `value = key`, in which case your delimiter is `"="`; or you may have comma-separated values in a data file.

The odds are good that you'll be doing two levels of splitting, like splitting by newlines, then splitting each line at the =. Two levels of breaking at delimiters is about enough to get you through just about anything short of writing a new programming language.

The standard C library since forever includes `strtok` (string tokenize). Its basic working is to step through the string you input until it hits the first delimiter, and overwrites the delimiter with a `'\0'`. Now the first part of the input string is a valid string representing the first token, and `strtok` returns the beginning of that substring for your use. The function holds the rest of the string internally, so when you call `strtok` again, it can search for the end of the next token, nullify that end, and return the head of that token as a valid string.

The function makes good use of how C works: the head of each token is just a pointer within an already-allocated string, and the tail is marked by a null character, so the tokenizing does a minimum of data copying and writing. The immediate implication is that the string you input is mangled, and because substrings are pointers, you can't free the input string until you are done using the substrings (or, you can use `strdup()` to copy out the substrings as they come out).

The `strtok` function holds the rest of the string you first input in a single static internal pointer, meaning that it is limited to tokenizing one string (with one set of delimiters) at a time, and it can't be used while threading. Therefore, consider `strtok` to be deprecated.

Instead, `strtok_r` is the reentrant version of `strtok`, and is what you should use. It is POSIX-standard, not C-standard. [The C11-standard version is covered below.] The use is a little awkward, because the first call is different from the subsequent calls.

- The first time you call the function, send in the string to be parsed as the first argument.
- On subsequent calls, send in `NULL` as the first argument.
- The last argument is the scratch string. You don't have to initialize it on first use; on subsequent uses it will hold the string as parsed so far.

Here's a line counter for you. Tokenizing is often a one-liner in scripting languages, but this is about as brief as it gets with `strtok_r`. Notice the `if ? then : else` to send in the original string only on the first use.

```
#include <string.h> //strtok_r
int count_lines(char *instring){
    int counter = 0;
    char *scratch, *txt, *delimiter = "\n";
    while ((txt = strtok_r((counter==0) ? instring : NULL, delimiter, &scratch))
        counter++;
    return out;
}
```

If you'd like a full example, have a look at the use of `strtok_r` in the Cetology example of Tip #52 (Entry #102).

If you want to be C11-standard instead of POSIX-standard, then use `strtok_s`, which works just like `strtok_r`, but has an extra argument (the second) which gives the length of the input string, and is updated to shrink to the length of the remaining string on each call. I suppose if the input string is not `\0`-delimited this extra element would be useful. We could redo the above example with:

```
#include <string.h> //strtok_s

//first use
size_t len = strlen(instring);
txt = strtok_s(instring, &len, delimiter, &scratch);

//subsequent use:
txt = strtok_s(NULL, &len, delimiter, &scratch);
```

To do:

Write yourself a function to take in a string and a delimiter list, and return a list of strings. Remember, you've got pointers to the text in the original string, so you don't need to allocate space for anything.