

Re: Reading a table

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2007-11/msg04506.html

- *From:* "Bill Reid" <hormelfree@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 30 Nov 2007 15:18:54 GMT
-

CBFalconer <cbfalconer@xxxxxxxx> wrote in message
<news:474F95D6.ECFC9965@xxxxxxxx>

Bill Reid wrote:

Roland Pibinger <rpbg123@xxxxxxxx> wrote in message

... snip ...

– without static FILE

Possibly, but the idea here is just to read a single file and then close it right up...

– for arbitrary long lines (no hard-coded maximum line length)

Yes, but again the idea here is we KNOW the maximum size of our file lines,

However, some people want to copy files without worrying about line length. With ggets (and fggets) you can handle this with ease, as in the following:

```
[1] c:\c\junk>cc -o fcopylns.exe ggets.o junk.c
```

```
[1] c:\c\junk>fcopylns <junk.c  
#include <stdio.h>  
#include <stdlib.h>  
#include "ggets.h"
```

Re: Reading a table

```
int main(void) {
char *line;

while (0 == ggets(&line)) {
puts(line);
free(line);
}
return 0;
} /* main, fcopylns */
```

Note the complexity. You can get the source etc. for ggets at:

<http://cbfalconer.home.att.net/download/ggets.zip>

Sure, something like that might come in handy in some situations, but I actually take a somewhat different tack for parsing out text "tables" with potentially (unpredictably) HUGE "field" sizes.

Consider that the file must STILL have a rigorously-applied delimiter strategy in order to be useful. Then consider that we can always replace the delimiter with '\0' in the text buffer, and that we can count the "lines" (rows) as being every occurrence of a '\n' OUTSIDE of a potentially HUGE text field "protected" in some way to allow the use of the "field" (columns) and "line" (rows) delimiters in the text.

So at the very least, if you use fgetc() until EOF, you will wind up with a buffer of strings delimited by NULs that corresponds to your count of "columns" and "rows", and if you haven't already completely parsed out the data and assigned it to your arrays, you can make a second pass to do so based on this information.

If you're wondering why you would want a second pass in the first place, I actually do this on a weekly basis with some HUGE text files (well, several MB each) I download from the net, and I actually prefer for download speed and security to go through the download buffers in real-time applying the NUL to each "field", and then after closing each download connection doing some fairly time-consuming parsing of the fields (which involves writing thousands of files).

In this case, I'm always doing dynamic memory re-allocation for the text buffer as the download buffer is filled on successive HTTP GETs; in the case of local files, a similar type of dynamic buffer re-sizing could also be used. Therefore, I never need to "get" a "line" of any length at all...

But when it comes time to parse a file that I've written myself and KNOW exactly how big the maximum line length is, I just use fgets(), and typically nothing more fancy than sscanf() to parse out and assign the "fields" for each line...

Re: Reading a table

William Ernest Reid