The simplest way to state this assignment is that the program is to read lines containing two fields, a file name and a data field. The program should append a line containing the data field to the file. It sounds simple, but it's not -- there are all sorts of crazy requirements. Remember how I said the first assignment was not about parsing input lines? This one is -- and the essential part of the assignment is properly sanitizing the inputs.

Restrictions:

There is no limit to the length of an input line, a data field, or a file name. It's acceptable for the program to abort if malloc() or new fail (though your program has to detect that and print an appropriate error message).

It is not acceptable to use libraries or other tools to aid in parsing. You may not use regular expression libraries and the like, and you may not use tools like flex. I want you to do it by hand. You may use the ctype functions or equivalent to determine the type of a given character.

All characters except NUL -- a byte composed of all zero bits -- are legal in data fields. All characters except NUL are acceptable in file names. That specifically means that characters with values 128-255 are legal; see https://en.wikipedia.org/wiki/Windows-1252 for a code chart.

Both file name (the first field) and the data name can be a string of letters and digits OR a quoted string. Letters and digits are as definited in that Web page. This includes the "international" letters (e.g., 0xC0-0xCF and others with that color background) and the superscript digits 0xB2, 0xB3, and 0xB9.

The two fields are separated by one or more blanks or tabs.

A quoted string starts with either a ' or a ", and ends with the same character. To include a quote mark inside a string, precede it with \; to include a \, write \\.

Strings can contain the C escapes \n, \t, and \r for Newline, Tab, and Return. They can also contain \ddd where 'd' is an octal digit. (Again, this is like C.) You must substitute the appropriate actual character for the special sequence.

These are legal strings:

```
Abç             (The last letter is 0xE7)
"Ab\347"        (This translates to the same thing!)
"Ab\\\n\r\""
" $%^&*( "
'"'
```

But "a\000b" is illegal, because NUL bytes are excluded.

These are, by intent, not exactly the parsing rules of either C or the shell.

The ultimate file name MUST be in either the current directory or /tmp. You MUST handle the .. problem. The program should append ".uni" to the filename, i.e., for me it would append .smb2132. The purpose of this rule is to prevent collisions by two people testing on the same

CLIC machine. (You do not need your VMs for this assignment.)

A file name like "/tmp/a/../foo" is legal (even if 'a' doesn't exist), because the resulting file is /tmp/foo. /tmp/a/b is illegal, even if /tmp/a exists; it's not in /tmp. /tmp/../tmp/foo is legal.

One last complication: you MUST do the actual file operation by invoking the system() library routine:

```
system("echo datafield >>filename");
```

That uses the shell, which means that after you parse the file name and the data field, you must properly escape them before invoking the shell. Why? Ask Little Bobby Tables....