CSC 209 Assignment 2, Fall 2013: C programming

Due by the end of Monday October 28, 2013; no late assignments without written explanation.

Part 1: Creating HTML <table> code

Manually writing fully-bracketed HTML code to create "tables" is a pain. A table which looks like this:

Assignment	**Weight**	**Out of**
1	5	10
2	5	30
3	10	20

is created by HTML code which looks something like this:

```
<table>
<tr><th>Assignment</th><th>Weight</th><th>Out of</th></tr>
<tr><td>1</td><td>5</td><td>10</td></tr>
<tr><td>2</td><td>5</td><td>30</td></tr>
<tr><td>3</td><td>10</td><td>20</td></tr>
</table>
```

Each table row is delineated by <tr>...</tr>; and each entry within the row is delineated by <td>...</td>, unless it is a heading, in which case we use <th> instead of <td>.

In this portion of the assignment you will write a tool named "table" to assist with this. I generated the contents of the above by passing four non-HTML-encoded text lines through "table –h1" (I separately typed the <table> and </table>).

Write a standard unix filter, which processes its standard input if there are no arguments, or processes a sequence of one or more file names on the command-line.

The transformation on the file(s) will be to enclose every line within <tr>...</tr> tags, and to divide every line up in accordance with tab characters present in the line and use either <td> or <th> as appropriate. So for example, I typed the first data line above as 1, tab, 5, tab, 10; passing this line through the "table" filter yields "<tr><td>1</td><td>5</td><td>10</td></tr>".

There is an optional argument "–h num" which states how many lines at the beginning are "heading" lines; if this option is not present, it defaults to zero. You must use getopt() in the standard way to parse the command-line options.

When a file cannot be opened, you must call perror() properly to output a standard error message.

Part 2: 'touch'

The *touch* command updates the mtime on a file to the current time. If the file does not exist, it is created. You will write this tool. Call your program "mytouch" to avoid confusion with /usr/bin/touch.

One common use of this tool is in dealing with automated tools which only process a file if it is newer than something else.

You will be writing the traditional 'touch' command, which does its work via file operations (rather than using utimes()). The algorithm is to open the file in "update" mode (i.e. for read and write, and without the truncate option so that the existing file contents are preserved); read one byte from the file; go back to the beginning of the file; and write that byte back to the file.

To perform these operations, use the low-level unix file functions: open, close, read, write, lseek, ftruncate. All of these file-handling functions deal with integer file descriptors (not FILE*). Your open() call will have three parameters: first the file name; then the middle parameter will be `O_RDWR|O_CREAT` which options say to open the file for update (read/write) and to create it if it doesn't exist; and the third parameter will be 0666 (the use of this value will be explained in lecture).

There are three cases you need to handle. First, the common case: if the file exists and is non-empty, you read one byte, call lseek(fd,0,SEEK_SET) to go back to the beginning, and write that byte. The second case is where the file doesn't already exist: your open() call will create it and that's good enough (since the newly-created file will have the current time as mtime); but most likely your program will not notice this situation and will treat it the same as the empty file case.

(over)

The third case, where the file is empty, is the only case in which we need the ftruncate() function. If the read() tells you you're at the end of the file (right away; i.e. the file is empty), you write an arbitrary byte to the file, then truncate the file to zero size with ftruncate(fd,0).

Your 'mytouch' command takes any number of file names as arguments. You do not need to use getopt(), but you must appropriately diagnose a usage error if there are no command-line arguments at all.

Part 3: Recursive directory listing

In this part of the assignment you will write a C program which is like an extremely simple version of 'find': It just lists the path names of the files in the specified directories and all subdirectories. It will be called "rls"; so "rls x y z" has the same output as "find x y z –print".

Furthermore, "rls" with no arguments will print from the current directory, with a difference to "rls .", illustrated as follows for hypothetical directory contents:

```
$ rls .
./foo
./bar
./baz
./baz/other
$ rls
foo
bar
baz
baz/other
$
```

You can omit printing the directory "." itself if you like (as I have above), and in any case you will skip "..". And be sure to use lstat() rather than stat().

Other notes

Your C programs must be in standard C. They must compile on the CDF machines with "gcc –Wall" with no errors or warning messages, and may not use linux-specific or GNU-specific features.

Pay attention to process exit statuses. Your programs must return exit status 0 or 1 as appropriate.

You should call your assignment files table.c, mytouch.c, and rls.c. Auxiliary files are not permitted. Once you are satisfied with your files, you can submit them for grading with the command

```
submit –c csc209h –a a2 table.c mytouch.c rls.c
```

and other submit commands are as in assignment one and the labs. This is the only submission method; you do not turn in any paper.

Please see the assignment Q&A web page at

```
http://www.cdf.toronto.edu/~ajr/209/a2/qna.html
```

for other reminders, and answers to common questions.

Remember:

This assignment is due at the end of Monday, October 28, by midnight. Late assignments are not ordinarily accepted and *always* require a written explanation. If you are not finished your assignment by the submission deadline, you should just submit what you have, for partial marks.

Despite the above, I'd like to be clear that if there *is* a legitimate reason for lateness, please do submit your assignment late and send me that written explanation.

And above all: Do not commit an academic offence. Your work must be your own. Do not look at other students' assignments, and do not show your assignment (complete or partial) to other students. Collaborate with other students only on material which is not being submitted for course credit.