CSC 209 Assignment 2, Winter 2016

Due by the end of Wednesday February 24, 2016; no late assignments without written explanation.

Part 1: ‘which’

Write the *which* command in C. For each of its arguments, the *which* program looks through your PATH and figures out which program the shell would run if you typed that command. For example, “which echo” will output “/bin/echo”.

Your program will be called “mywhich”. To simplify the string handling, your program will not actually use the PATH environment variable but rather will have a built in search list which is: /bin; then /usr/bin; then the current directory. Your program’s output will tell the user the match which occurs *first*; the search list is in order (like the PATH search list is).

The real ‘which’ command checks whether the files are executable by you. For the purposes of this assignment you will more simply instead check if the files are executable by owner. Note that, for example, if there is a non-executable file /bin/blof and an executable file /usr/bin/blof, “which blof” will output “/usr/bin/blof”. Whereas if /bin/blof were executable, “which blof” would output “/bin/blof”.

If an executable file by that name is not found in the search list, output an error message of the form “whatever: Command not found.”. (The period at the end of this output is odd for unix, but is traditional for ‘which’.)

“mywhich” with no command-line arguments should generate a usage message in the standard format (resembling the SYNOPSIS line in a man page).

Part 2: ‘uniq’

Write a standard unix tool which is a simplified *uniq*: its only command-line option is −c (use getopt() to parse the command-line options). Unlike the real ‘uniq’, but like most standard unix tools, it takes zero or more file names on the command-line in the usual way.

Call your program “myuniq”.

Entire lines are compared for uniqueness, so you can just “fgets” into two alternating buffers. You can fgets() into a buffer of size 500 bytes, such that lines longer than 499 bytes (including the newline) will be broken; as always there must not exist any possible input which would cause your program to exceed any array bounds.

Part 3: ‘du’

The “du” command recursively lists disk usage under a specified directory. The “st_blocks” value in the struct filled up by lstat() will tell you the number of blocks used by a particular file times two (that is, it assumes that blocks are half-K).

Your “mydu” program will have command-line options ‘−h’, which means to use suffixes K, M, and G as applicable (e.g. 1132 blocks would instead be output as “1M”); and ‘−s’, which means to output only the summary line for each argument (by default it outputs a line for each subdirectory as it traverses). Use getopt() to parse the command-line options. Having no filename arguments is a usage error.

Note that directories’ own disk usage does not contribute to the count; add st_blocks/2 only for non-directories. Also note that it is important to use lstat() rather than stat() to avoid the possibility of infinite recursion.

The size is separated from the file name simply by a single space (unlike the cuter formatting performed by the “real” *du*). Also, your mydu program will expect the command-line arguments to be directories (whereas du itself allows plain files as well).

mydu should *not* do a chdir() (which you might be tempted to do to avoid string processing). If it does a chdir(), your program will no longer always work with multiple command-line arguments, since a chdir() in processing the first directory may invalidate the name of the specified second directory.

You may assume a maximum likely complete path name of, say, 2000 characters, so long as your program aborts with an error, rather than exceeding array bounds, should the situation turn out to be more complex than this.

You may not use ftw() or fts().

(over)

Other notes

You can find compiled solutions for all of these programs in /student/csc209/a2 for comparison.

Your C programs must be in standard C. They must compile on the UTM linux 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.

Call your assignment files mywhich.c, myuniq.c, and mydu.c. Your files must have the correct names to be processed correctly by the grading software, and auxiliary files are not permitted.

Once you are satisfied with your files, you can submit them for grading with the command

```
/student/csc209/submit a2 mywhich.c myuniq.c mydu.c
```

You may still change your files and resubmit them any time up to the due time. You can check that your assignment has been submitted with the command

```
/student/csc209/submit -l a2
```

This is the only submission method; you do not turn in any paper.

Please see the assignment Q&A web page at

```
https://cs.utm.utoronto.ca/~ajr/209/a2/qna.html
```

for other reminders, and answers to common questions.

Remember:

This assignment is due at the end of Wednesday, February 24, 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. (For medical notes, I need to see the original, in person. E.g. in office hours or after the next class.)

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.