CSC 352, Fall 2015
Assignment 3
Due: Wednesday, September 16 at 23:59:59

Introduction

The a2 write-up started with 3+ pages of various information about assignments. None of that is repeated here, but all of it applies this assignment, too.

You'll need to make an a3 symlink instead of an a2 symlink, of course.

Problem 1. (10 points) clp.java

It's been said that you never really understand how a piece of software works until you have to implement it. For this problem you are to implement a parser in Java for very, very simple bash command lines read from standard input. The parser output identifies commands, their arguments, and what the command uses for standard input and standard output.

Here is a simple example:

```
% java clp
wc --lines x y | cat -n > out
----------------------------------------
Command line: wc --lines x y | cat -n > out

Command 1: wc
    Arguments:
        |--lines|
        |x|
        |y|
    Standard input:  [keyboard]
    Standard output: [pipe]

Command 2: cat
    Arguments:
        |-n|
    Standard input:  [pipe]
    Standard output: out
```

Let's walk through what happened. The user typed "wc --lines x y | cat -n > out" and hit ENTER. The program then printed a line of dashes followed by the line the user entered. (We can say clp "echoes the input".)

clp then went command-by-command, printing the command, the arguments, and an indication of what is specified or implied for standard input and standard output. [keyboard] and [pipe] are enclosed in brackets to distinguish them from a file name; [screen], shown below, is similar.

Here are the rules for the very, very simple command lines that clp.java can handle.

- The only metacharacters recognized are <, > and |, and they may only be used as individual characters. (The sequence >> is not permitted, for example.)
- Uses of <, > and | must be surrounded by at least one space. < and > may also start a line.

Note the spaces that surround `|` and > in this line:

```
      wc --lines x y | cat -n > out
```

<u>bash does not require those spaces but `clp` does!</u> Here's an invalid command line for `clp`:

```
      wc --lines x y|cat -n>out
```

It's invalid because the two operators aren't surrounded by spaces.

Here's a command line that you may first think is invalid <u>**but it is valid**</u>:

```
      % java clp
      ls-t > x wc-l
      ----------------------------------------
      Command line: ls-t > x wc-l

      Command 1: ls-t
          Arguments:
              |wc-l|
          Standard input:  [keyboard]
          Standard output: x
```

You might think that the user forgot a space between "`ls`" and "`-t`" and maybe meant to pipe into `wc -l` but messed it up. We see that `clp` considers the command to be `ls-t`, and considers `wc-l` to be an argument. <u>That's exactly how bash would view that line, too.</u> Don't imagine that `clp` needs to do any human-like reasoning; it doesn't, and neither does bash.

`clp` reads and processes lines until end of file is reached. Example:

```
      % java clp
      cat
      ----------------------------------------
      Command line: cat

      Command 1: cat
          No arguments
          Standard input:  [keyboard]
          Standard output: [screen]
      ls -l -t -r > out
      ----------------------------------------
      Command line: ls -l -t -r > out

      Command 1: ls
          Arguments:
              |-l|
              |-t|
              |-r|
          Standard input:  [keyboard]
          Standard output: out
      ^D  (control-D)
      %
```

<u>**`clp` does no error checking!**</u> <u>**Behavior is undefined**</u> for lines like "`a || b`", "`| x |`", "`ls >> x`", and infinitely more!

Behavior is undefined for commands that direct standard output to both a pipe and a file, like "`ls > x | wc`", or multiple files, like "`date > x > y`". Ditto for standard input ambiguity with something like "`cal | x < y`" or "`wc < x < y`".

Behavior is undefined if a command line is empty or contains only spaces.

For more examples of `clp` input and resulting output, see `a3/master/tester.out/clp.out.*`.

Beat-the-Coach time: 62 minutes, 49 seconds. No IDE.

Stupid questions

You've perhaps heard teachers say, "There is no such thing as a stupid question." Now that you're sophomores or better, I must tell you that is incorrect. There are lots of stupid questions! Here are examples of questions about `clp` that I would consider to be stupid questions:

"Should my version output the same number of '-'s as yours does?"
Yes, if you want to pass any tests.

"Does capitalization matter?"
Yes, if you want to pass any tests.

It looks like you've got two spaces after "Standard input:" but only one space after "Standard output:". Should I fix that?
No, not unless you want to fail every test.

Implementation notes

Some of you may see the word "parse" above and soon be asking questions about ANTLR, but this is a simple problem of chopping up a line into "words" and then working through those words. A simple `String.split()` with a regular expression should be all you need to get started. Assuming that `line`, of type `String`, is the command line being parsed, here's how to split the line into words:

```
String words[] = line.split("\\s+");
```

Various levels of hints can be found in `a3/clp-hints*`. Start with `clp-hints`, after rereading the section on hints in the `a2` write-up.

Problems 2-4. (2 points each; 6 points total) `tree-patrick`, `tree-praharsh`, and `tree-youhao`

Last week I asked each of the TAs to create some tree navigation problems like we worked through on slide 121. Here's the model I gave them to work from:

```
% cat a3/tree-whm
mkdir whm; cd whm
mkdir -p 352/{d1,work}
touch 352/{one,two}
cd 352/d1
mkdir -p notes/{platforms,langs}
cal >cal.out
touch notes/langs/java

exit
```

```
Q: In work, how can I cd to 352?

Q: In work, how can I cd to d1?

Q: In platforms, how can I cat two?

Q: In d1, does 'cd ../d1' work?

Q: In d1, does 'cat d1/cal.out' work?
```

Note that the file starts with a number of commands that build a tree in the current directory. It then uses `exit` to terminate the script. The questions on the lines that follow are ignored.

Before we run it, I need to mention a bash feature we haven't seen before: *brace expansion*. Here's the first example of it in `a3/tree-whm`:

```
mkdir -p 352/{d1,work}
```

Let's use `echo` to see what that means:

```
% echo mkdir -p 352/{d1,work}
mkdir -p 352/d1 352/work
```

Here's a more complex example:

```
% echo {a,b/c,d}/{one,two}
a/one a/two b/c/one b/c/two d/one d/two
```

My 2005 UNIX slides, on Piazza, have more examples of brace expansion on 72-74.

I won't expect any detailed knowledge of brace expansion on quizzes or exams, but I will use brace expansions from time to time for concise expression.

Let's run `a3/tree-whm` to create the tree we need to answer the questions. We could make that file executable with `chmod` but since we just need to run it once, let's just tell bash to run it:

```
~/352/a3 % ls
~/352/a3 % bash a3/tree-whm
~/352/a3/x % ls
whm
```

Let's look around in the tree that was made. `find` is used below but try `ls -lR`, too.

```
~/352/a3 % cd whm
~/352/a3/whm % find
.
./352
./352/d1
./352/d1/notes
./352/d1/notes/platforms
./352/d1/notes/langs
./352/d1/notes/langs/java
./352/d1/cal.out
./352/work
./352/one
```

```
./352/two
```

With the all the above in hand, here's the task for you:

1. Copy `a3/tree-{patrick,praharsh,youhao}` into your directory:
   ```
   ~/352/a3 % cp ../a3/tree-{patrick,praharsh,youhao} .
   ```

2. Run each one, to create the tree it uses:
   ```
   ~/352/a3 % for i in tree-*; do bash $i; done
   ```

3. Edit the three `tree-*` files, adding your answers on a line immediately following each question, but don't change or delete any existing lines. Don't prefix your answers with "`A:`", a prompt, or anything else.

   Using `a3/tree-whm` as an example, here's how a completed file should look:

   ```
   % cat tree-whm
   mkdir whm; cd whm
   ...more lines unchanged...

   exit
   Q: In work, how can I cd to 352?
   cd ..

   Q: In work, how can I cd to d1?
   cd ../d1

   Q: In platforms, how can I cat two?
   cat ../../../two

   Q: In d1, does 'cd ../d1' work?
   yes

   Q: In d1, does 'cat d1/cal.out' work?
   no
   ```

A "diff" should show nothing but line number information and your answer lines, something like this, but you'll have ten answers in each file.

```
~/352/a3 % diff a3 tree-whm
11a12
> cd ..
13a15
> cd ../d1
15a18
> cat ../../../two
17a21
> yes
19a24
> no
```

Let's discard that line number information:

```
% diff a3 tree-whm | grep ">"
> cd ..
> cd ../d1
```

```
> cat ../../../two
> yes
> no
```

I apologize for all the tedious detail on this one—I'm trying to enable some speedy grading of the 4,320 answers we'll be dealing with! :)

Remember: You're filling in answers in three files: `tree-patrick`, `tree-praharsh`, and `tree-youhao`.

Problem 5. (3 points) `getrc`

Following the rules for one-pipeline bash scripts in the `a2` write-up, write a one-pipeline bash script `getrc` that outputs the value at a specified row and column position in a file whose lines consist of comma-separated values.

Here's a file that we'll view as having three rows and two columns:

```
% cat a2/getrc.1
a,b
cc,d
e,fff
```

Let's use `getrc` to fetch some values:

```
% getrc 1,1 a3/getrc.1
a
% getrc 2,2 a3/getrc.1
d
% getrc 3,2 a3/getrc.1
fff
```

Note that the row and column specification is a single argument.

Assume all rows are well formed. You won't see a row like "`a,,b`" or "`a,b,c,`".

Behavior is undefined if a specified row or column does not exist.

Before you go digging around in the bash man page for string manipulation operators and such let me remind you that as usual, what you've seen on the slides and in previous assignments is all you need to solve the problem.

FYI, Files like `getrc.1` are known as CSV files but the general format is more complex. For example, Excel might produce a CSV file with a four-value line that contains quotes, like this:

```
1,2,"3,a,test",4
```

You won't see anything like the line above in the tests for `getrc`.

Problem 6. (2 points) `hgram`

Write a bash script `hgram` that reads non-negative integers from standard input, zero or more per line, and outputs a simple histogram of the values.

```
% echo 5 2 0 3 | hgram
*****
**

***
% cal -h 9 2015 | tail -3 | tac | hgram
***************************
****************************
*****************************
******************************
********************
*********************
**********************
***********************
************************
*************************
**************************
% hgram < /dev/null | wc -c
0
```

Unlike all the scripts you've written to date, `hgram` isn't required to be a one-liner. My solution starts with these two lines:

```
for value in $(cat)
do
```

Problem 7. `questions.txt`

`a3/questions.txt` is a plain text file with a number of free-response questions. Copy the file into your directory and edit your answers into it, leaving the questions intact. Add your answers after the "`Answer:`" lines. Answers may be multi-line. Per-problem points are specified in the file.

If you're unsure about the format for any of your answers, mail it to `352f15` and we'll take a look!

Problem 8. Extra Credit `observations.txt`

Submit a plain text file named `observations.txt` with...

(a) (1 point extra credit) An estimate of how long it took you to complete this assignment. To facilitate programmatic extraction of the hours from all submissions have an estimate of hours on a line by itself, more or less like one of the following three examples:

```
Hours: 6
Hours: 3-4.5
Hours: ~8
```

If you want the one-point bonus, be sure to report your total (estimated) hours on a line that starts with "`Hours:`".

Other comments about the assignment are welcome, too. Was it too long, too hard, too detailed? Speak up! I appreciate all feedback, favorable or not.

(b) (1-3 points extra credit) Cite an interesting course-related observation (or observations) that you made while working on the assignment. The observation should have at least a little bit of depth. Think of me saying "Good!" as one point, "Excellent!" as two points, and "Wow!" as three points. I'm looking for quality, not quantity.

Turning in your work

Use `a3/turnin` to submit your work. It creates a time-stamped "tar file" of your work. You can run it as often as you want. We'll grade your final submission.

`a3/turnin -l` shows your submissions.

To give you an idea about the sizes of my solutions, here's what I see as of press time.

```
% wc $(cat a3/delivs)
  92  256 2327 clp.java
  31  117  637 tree-patrick
  38  148  859 tree-praharsh
  41  170 1462 tree-youhao
   1   20   82 getrc
   8   17   88 hgram
  84  477 3451 questions.txt
   0    0    0 observations.txt
 295 1205 8906 total
```

There are no comments in my code. My `questions.txt` and `tree-*` files have not been populated with answers yet.

Miscellaneous

This assignment is based on the material on UNIX slides 1-177.

Point values of problems correspond directly to assignment points in the syllabus. For example, a 10-point problem would correspond to 1% of your final grade in the course.

Feel free to use comments to document your code as you see fit, but note that no comments are required, and no points will be awarded for documentation itself. (In other words, no part of your score will be based on documentation.)

Remember that late assignments are not accepted and that there are no late days; but if circumstances beyond your control interfere with your work on this assignment, there may be grounds for an extension. See the syllabus for details.

My estimate is that a student who has only taken CSC 127A and 127B but done well in them and has completed the previous assignments will need 6-8 hours to complete this assignment.

Keep in mind the point value of each problem; don't invest an inordinate amount of time in a problem or become incredibly frustrated before you ask for a hint or help. Remember that the purpose of the assignments is to build understanding of the course material by applying it to solve problems. If you reach the six-hour mark, regardless of whether you have specific questions, it's probably time to touch base with us. Give us a chance to speed you up! **Our goal is that everybody gets 100% on this assignment AND gets it done in an amount of time that is reasonable for them.**

I hate to have to mention it but keep in mind that cheaters don't get a second chance. If you give your code to somebody else and they turn it in, you'll both likely fail the class, and more. (See the syllabus for the details.)