CSC 209 Assignment 1, Summer 2022: Shell programming

Due by the end of Monday June 6, 2022; no late assignments without written explanation.
Please re-read the statement about academic offences at the end of the course information sheet!

1. rmlink

First, consider a shell script which goes through its command-line arguments and removes those files if and only if they are symbolic links (`test -L`). Files which are not symbolic links are silently ignored.

Your program should behave like a standard unix tool; it makes no sense to run this program with no arguments, so that should yield a usage error.

The simplest case

```
sh rmlink file
```

doesn't have many applications; but a commoner use is something like

```
sh rmlink *
```

which deletes all symlinks in the current directory (excluding files whose names begin with a dot), but leaves all other files alone.
(Of course, your program will work in either case.)

However, to avoid various kinds of accidents in development, this assignment question actually asks you to run "`ls -l`" on each of the symbolic links, rather than "`rm`".

2. dist

The program "dist" will read a sequence of integers, and will determine the adjacent pair with the largest numeric distance between them (i.e. abs(x−y)). It outputs three numbers: the distance plus the two numbers exhibiting that distance.

For example, if the input numbers are `3 10 2 -5`
then the output would be `8 10 2`
because 10 to 2 is the largest distance between adjacent numbers (slightly beating out the distance from 3 to 10). The distance from 10 to −5 doesn't count because they aren't adjacent.

So you don't need to examine all of the numbers at once; just a current number and a previous number, as you go around your loop.

Your program will read numbers from files, not from the command-line: The arguments on the command-line are file names, and each file is consulted for a string of numbers. If there are no arguments on the command-line, then it reads the standard input, like a normal unix tool.

The easiest way to accomplish this is to use `cat "$@"`

This command will process files in exactly the way you need. You don't need to look at the files in any other way, nor to make your own decision about whether to read file names or the standard input — use `cat` to do this.

Numbers may be separated by spaces or newlines. Blank lines and repeated spaces are allowed.

If the input contains only whitespace or only one number, there should be no output.

Note that you can take the absolute value of an integer by passing it through "`tr -d -`".

3. adv (adventure)

You will write a processor for data representing a "choose your own adventure" story.

This might best be illustrated by the distributed sample database. If you run "`sh adv /u/csc209h/summer/pub/a1/advdb`", the output should at first look like this:

```
Class is over and you are hungry.  You have money for lunch.
Do you
    1) leave the classroom in search of food
    2) remain seated
?
```

and then it will wait for input.

If you then type "1", it will say:

```
You seem to be in a very different building from the one you thought you
entered the classroom from.  There are three doors, unmarked.  You can't
tell which one is the classroom you just came out of; you aren't sure
that it's any of the three.
Do you
    1) go through the door on the left
    2) go through the door in the middle
    3) go through the door on the right
?
```

and so on.

The database argument is expected to be a directory. It should contain a file named "story". This file is output for that portion of the story.
The directory can also contain numeric subdirectories named 1, 2, 3, and so on, up to a maximum of 9. There is no minimum. If even the subdirectory "1" doesn't exist, the game is over and the program exits. Otherwise, it outputs "Do you", the selections based on the available subdirectories, and the question mark. Each subdirectory contains a file "choice" describing that choice.

Once a choice has been made, the program continues with that subdirectory. That is to say, that subdirectory will (also) contain a file named "story", and possibly numeric subdirectories as above.

Please see an example database at `/u/csc209h/summer/pub/a1/advdb` .
For further elucidation, please try a sample solution which is written in C instead of *sh*, so that I can give you an executable without showing you the program, at `/u/csc209h/summer/pub/a1/adv` .

To submit

I suggest you begin by making a new directory to hold your assignment files, plus any other working

copies and associated files. Your programs must have the names indicated above.

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

```
submit -c csc209h -a a1 rmlink dist adv
```

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

```
submit -l -c csc209h -a a1
```

You can also submit files individually instead of all at once, and you can resubmit a particular file by using the –f option, e.g.

```
submit -c csc209h -a a1 -f rmlink
```

Other notes

Your program must run on the teaching lab linux machines, and it should use standard *sh* features only (avoiding *bash* extensions). Please test it in different shells, not only in *bash*.
E.g. also try "`dash adv /u/csc209h/summer/pub/a1/advdb`".

The commonest ways which people in CSC 209 at this point use *bash* extensions is to use the built-in arithmetic functions, or to use arrays. **Don't do this. You will lose marks.** Develop your program so that it works under both *bash* and *dash*, especially *dash*.

For now, we will run our shell scripts by typing "sh file" or "sh file args ...".

We will not be worrying about the "#!/bin/sh" thing for assignment one. (We'll talk about this later in the course.)

Do not use python, awk, perl, or any other programming language in your scripts besides *sh*. These are valuable languages to know (well, at least python and awk are), but are not the point of the current assignment, in which you will be graded on your *sh* programming.

Please see the assignment Q&A web page at https://www.teach.cs.toronto.edu/~ajr/209/a1/qna.html for other reminders, advice, and answers to common questions.

Make your program look reasonably nice. For example, standard indentation is required. But avoid clutter (e.g. banner comments, or comments which attempt to teach the reader the *sh* programming language). (We do realize that programs in *sh* rarely look as clear as programs in a normal programming language.)

Remember:

This assignment is due at the end of Monday, June 6, 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. Please see
https://www.teach.cs.toronto.edu/~ajr/209/specialcons.html

I'd also like to point out that even a zero out of 8% is far better than cheating and suffering an academic penalty. Don't cheat even if you're under pressure. Whatever the penalty eventually applied for cheating, it will be worse than merely a zero on the assignment. 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.