Objectives

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab04` by typing:

```
$ mkdir lab04
```

Change to this directory by typing:

```
$ cd lab04
```

Exercise: Deja Vu - Mapping Digits but in Perl (individual)

This is an individual exercise to complete by yourself.

Write a Perl script `digits.pl` that reads from standard input and writes to standard output mapping all digit characters whose values are less than 5 into the character '<' and all digit characters whose values are greater than 5 into the character '>'. The digit character '5' should be left unchanged.

Sample Input Data	Corresponding Output
`1 234 5 678 9`	`< <<< 5 >>> >`

`I can think of 100's`
`of other things I'd rather`
`be doing than these 3 questions`
	`I can think of <<<'s`
	`of other things I'd rather`
	`be doing than these < questions`

`A line with lots of numbers:`
`12345678912345678912345678 9`
`A line with all zeroes`
`00000000000000000000000000`
`A line with blanks at the end`
`1 2 3`
	`A line with lots of numbers:`
	`<<<<5>>>>><<<<5>>>>><<<<5>>>>`
	`A line with all zeroes`
	`<<<<<<<<<<<<<<<<<<<<<<<<<<`
	`A line with blanks at the end`
	`< < <`

`Input with absolutely 0 digits in it` `Well ... apart from that one ...`	`Input with absolutely < digits in it` `Well ... apart from that one ...`
`1 2 4 8 16 32 64 128 256 512 1024` `2048 4096 8192 16384 32768 65536`	`< < < > <> << >< <<> <5> 5<< <<<<` `<<<> <<>> ><>< <><>< <<>>> >55<>`

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest perl_digits
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab04_perl_digits digits.pl
```

You must run give before **Tuesday 02 July 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Exercise: Echoing A Shell Exercise in Perl? (individual)

This is an individual exercise to complete by yourself.

Write a Perl script `echon.pl` which given exactly two arguments, an integer *n* and a string, prints the string *n* times. For example:

```
$ ./echon.pl 5 hello
hello
hello
hello
hello
hello
$ ./echon.pl 0 nothing
$ ./echon.pl 1 goodbye
goodbye
```

Your script should print exactly the error message below if it is not given exactly 2 arguments:

```
$ ./echon.pl
Usage: ./echon.pl <number of lines> <string>
$ ./echon.pl 1 2 3
Usage: ./echon.pl <number of lines> <string>
```

Also get your script to print this error message if its first argument isn't a non-negative integer:

```
$ ./echon.pl hello world
./echon.pl: argument 1 must be a non-negative integer
$ ./echon.pl -42 lines
./echon.pl: argument 1 must be a non-negative integer
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest perl_echon
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab04_perl_echon echon.pl
```

You must run give before **Tuesday 02 July 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Exercise: A Perl Tail (individual)

This is an individual exercise to complete by yourself.

Perl file manipulation

The standard approach in Perl for dealing with a collection of files whose names are supplied as command line arguments, is something like:

```
#!/usr/bin/perl -w
foreach $arg (@ARGV) {
    if ($arg eq "--version") {
        print "$0: version 0.1\n";
        exit 0;
    # handle other options
    # ...
    } else {
        push @files, $arg;
    }
}

foreach $file (@files) {
    open F, '<', $file or die "$0: Can't open $file: $!\n";

    # process F

    close F;
}
```

Write a Perl script to implement the Unix `tail` command. It should support the following features of `tail`:

- read from files supplied as command line arguments
- read from standard input if no file name arguments are supplied
- display the error message `tail.pl: can't open` *`FileName`* for any unreadable file
- display the last *N* lines of each file (default *N* = 10)
- can adjust the number of lines displayed via an optional first argument *−N*
- if there is more than one named file, separate each by `==>` *`FileName`* `<==`

To assist with testing your solution, there are three small test files: t1.txt, t2.txt, and t3.txt. Copy these files to your current directory.

```
$ cp /web/cs2041/19T2/activities/perl_tail/t?.txt .
```

Using these data files, your program should behave as follows:

```
$ ./tail.pl <t1.txt
Data 1 ... Line 2
Data 1 ... Line 3
Data 1 ... Line 4
Data 1 ... Line 5
Data 1 ... Line 6
Data 1 ... Line 7
Data 1 ... Line 8
Data 1 ... Line 9
Data 1 ... Line 10
Data 1 ... Last line
$ ./tail.pl t1.txt
Data 1 ... Line 2
Data 1 ... Line 3
Data 1 ... Line 4
Data 1 ... Line 5
Data 1 ... Line 6
Data 1 ... Line 7
Data 1 ... Line 8
Data 1 ... Line 9
Data 1 ... Line 10
Data 1 ... Last line
$ ./tail.pl -5 t1.txt
Data 1 ... Line 7
Data 1 ... Line 8
Data 1 ... Line 9
Data 1 ... Line 10
Data 1 ... Last line
$ ./tail.pl -5 t2.txt
A one line file.
$ ./tail.pl -5 t1.txt t2.txt t3.txt
==> t1.txt <==
Data 1 ... Line 7
Data 1 ... Line 8
Data 1 ... Line 9
Data 1 ... Line 10
Data 1 ... Last line
==> t2.txt <==
A one line file.
==> t3.txt <==
one
word
on
each
line
$ ./tail.pl -2 tX.txt
./tail.pl: can't open tX.txt
```

Hint: use the above template for Perl file processing to get started with your script. You *must* use Perl's `-w` flag in your script, and you must write your code in such a way as to ensure that no warning messages are produced.

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest perl_tail
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab04_perl_tail tail.pl
```

You must run give before **Tuesday 02 July 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Challenge Exercise: Shuffling Lines (individual)

This is an individual exercise to complete by yourself.

Write a Perl script `shuffle.pl` which prints its input with the lines in random order. For example, lets create a file containing the integers 0..4.

```
$ seq 0 4 >numbers.txt
```

Now if we run `shuffle.pl` taking its input from this file it should print the lines in a different order each time its run, for example:

```
$ cat numbers.txt
0
1
2
3
4
$ ./shuffle.pl <numbers.txt
0
4
3
2
1
$ ./shuffle.pl <numbers.txt
4
3
0
1
2
$ ./shuffle.pl <numbers.txt
2
4
0
3
1
```

You are not permitted to use `List::Util` (it contains a shuffle function).
Don't look for other people's solutions - see if you can come up with your own. **Hint:** the perl function *rand* returns a floating point number between 0 and its argument. For example:

```
$ perl -e 'print rand(42), "\n"'
35.8075417916499
```

```
$ perl -e 'print rand(42), "\n"'
40.8355243656062
```

Hint: perl ignores the fractional part of a number if you use it to index an array There is no autotest and no automarking of this question.
When you have completed demonstrate your work to another student in your lab and ask them to enter a peer assessment here

It is prefered you do this during your lab, but if this is not possible you may demonstrate your work to any other COMP(2041|9044) student before Tuesday 02 July 17:59:59. Note, you must also submit the work with give.

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab04_perl_shuffle shuffle.pl
```

You must run give before **Tuesday 02 July 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Challenge Exercise: Testing a Non-determinate Program (individual)

This is an individual exercise to complete by yourself.

There is no dryrun test for `shuffle.pl`. Testing (pseudo)random programs is more difficult because there are multiple correct outputs for a given input.

Write a shell script `shuffle_test.sh` which tests `shuffle.pl`.

Try to test that all outputs are correct and all correct outputs are being generated.

There is no autotest and no automarking of this question.
When you have completed demonstrate your work to another student in your lab and ask them to enter a peer assessment here

It is prefered you do this during your lab, but if this is not possible you may demonstrate your work to any other COMP(2041|9044) student before Tuesday 02 July 17:59:59. Note, you must also submit the work with give.

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab04_shuffle_test shuffle_test.sh
```

You must run give before **Tuesday 02 July 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Submission

When you are finished each exercises make sure you submit your work by running **give**.
You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Tuesday 02 July 17:59:59** to submit your work.

You cannot obtain marks by e-mailing lab work to tutors or lecturers.

You check the files you have submitted here

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

(Hint: do your own testing as well as running `autotest`)

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface

Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

The lab exercises for each week are worth in total 1 mark.
The best 10 of your 11 lab marks will be summed to give you a mark out of 9. If their sum exceeds 9 - your mark will be capped at 9.

- You can obtain full marks for the labs without completing challenge exercises
- You can miss 1 lab without affecting your mark.

COMP(2041|9044) 19T2: Software Construction is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au

CRICOS Provider 00098G