You can do anything with Perl

1. Write a Perl program, `tags.pl` which given the URL of a web page fetches it by running *wget* and prints the HTML tags it uses.
   The tag should be converted to lower case and printed in sorted order with a count of often each is used.

   Don't count closing tags.

   Make sure you don't print tags within HTML comments.

   For example:

   ```
   $ ./tags.pl http://www.cse.unsw.edu.au
   a 141
   body 1
   br 14
   div 161
   em 3
   footer 1
   form 1
   h2 2
   h4 3
   h5 3
   head 1
   header 1
   hr 3
   html 1
   img 12
   input 5
   li 99
   link 3
   meta 4
   noscript 1
   p 18
   script 14
   small 3
   span 3
   strong 4
   title 1
   ul 25
   ```

   Note the counts in the above example will not be current - the CSE pages changes almost daily.

2. Add an -f option to tags.pl which indicated the tags are to printed in order of frequency.

```
$ tags.pl -f http://www.cse.unsw.edu.au
head 1
noscript 1
html 1
form 1
title 1
footer 1
header 1
body 1
h2 2
hr 3
h4 3
span 3
link 3
small 3
h5 3
em 3
meta 4
strong 4
input 5
img 12
br 14
script 14
p 18
ul 25
li 99
a 141
div 161
```

3. Write a Perl function which takes an integer argument n and reads the next n lines of input and returns them as a string.

4. Write a Perl program `./shpl.pl` which reads Shell and outputs Perl for example:

```
while ls important_file >/dev/null
do
        echo "all OK"
        sleep 1
done
echo "Panic important_file gone"
```

Your program should produce this output:

```
while (!system "ls important_file >/dev/null") {
        print "all OK\n";
        system("sleep 1");
}
print "Panic important_file gone\n";
```

Assume that the only Shell features that you need to handle specially are while loops & echo statements, and assume the Shell is nicely formatted. Don't worry about the "#!" line.
Write a quick version using regexes which does no checking just translates

Then write a longer version that checks the syntax of the Shell while loops are correct, including handling nested while loops.
Hint: use recursive functions.

5. Give Perl code which given the name of a C function searches the C source files (*.c) in the current directory for calls of the function, declarations & definitons of the function and prints a message indicating the file and line number, in the format below.
You can assume functions are defined with the type, name and paramaters on a single non-indented line. You can assume function bodies are always indented.

You don't have to handle multi line comments. Try to avoid matching the function name in strings or single line comments
For example:

```
$ cat half.c
double half(double x) {
    return  x/2;
}
$ cat main.c
#include <stdio.h>
#include <stdlib.h>

double half(double x);

int main(int argc, char *argv[]) {
    return  half(atoi(argv[1]));
}
$ ./print_function_uses.pl half
a.c:1 function half defined
half.c:1 function half defined
main.c:4 function half declared
main.c:7 function half used
```

6. Give Perl code which given a C program as input finds the definitions of single parameter functions and prints separately the function's type, name and the parameters name & type. Assume all these occur on a single non-indented line in the C source code. You can assume function bodies are always indented. Allow for white space occuring anywhere in the function header. You can assume that types in the program don't contain square or round brackets. For example:

```
$ cat a.c
double half(int *x) {
    return  *x/2.0;
}
$ ./print_function_types.pl a.c
function type='double'
function name='half'
parameter type='int *'
parameter name='x'
```

7. Write a Perl script C_include.pl which given the name of a C source file prints the file replacing any '#include' lines with the contents of the included file, if the included file itself contains a #include line these should also be processed.
   Assume the source files contain only quoted ("") include directives which contain the files's actual path name. For example:

```
$ cat f.c
#include "true.h"

int main(int argc, char *argv[]) {
    return  TRUE || FALSE;
}
$ cat true.h
#define TRUE 1
#include "false.h"
$ cat false.h
#define FALSE 0
$ ./C_include.pl f.c
#define TRUE 1
#define FALSE 0

int main(int argc, char *argv[]) {
    return  TRUE || FALSE;
}
```

8. Modify C_include.pl so that it handles both `""` and `<>` directives. It should search the directories `/usr/include/`, `/usr/local/include` and / `usr/include/x86_64-linux-gnu` for include files specified in `<>` directives and for files specified in `""` directives which do not exist locally. For example:

```
$ cat g.c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("hello world\n");
    exit(0);
}
$ ./C_include.pl g.c
/* Define ISO C stdio on top of C++ iostreams.
   Copyright (C) 1991, 1994-2008, 2009, 2010 Free Software Foundatio
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/o
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version

   The GNU C Library is distributed in the hope that it will be usef
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, write to the Free
   Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
   02111-1307 USA.  */

/*
 *      ISO C99 Standard: 7.19 Input/output     <stdio.h>
 */

#ifndef _STDIO_H

#if !defined __need_FILE && !defined __need___FILE
# define _STDIO_H       1
/* Copyright (C) 1991-1993,1995-2007,2009,2010,2011
   Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/o
...
```

9. Write a Perl program, `times.pl` which prints a table of multiplications.
   Your program will be given the dimension of the table and the width of the columns to be printed. For example:

```
$ times.pl 4 5 3
  1   1   2   3   4   5
  2   2   4   6   8  10
  3   3   6   9  12  15
  4   4   8  12  16  20
```

10. Write a Perl program which deletes blank lines from each of the files specified as arguments. For example, if run like this:

```
$ deblank.pl file1 file2 file3
```

your program should delete any blank lines in `file1`, `file2` and `file3`. Note that this program *changes* the files, it doesn't just write the "de-blanked" versions to standard output.

11. Write a Perl function `listToHTML()` that given a list of values returns a string of HTML code as an unordered list:

```
out = listToHTML('The', 'Quick', 'Brown', 'Fox');
```

would result in `$out` having the value ...

```
<ul>
<li>The
<li>Quick
<li>Brown
<li>Fox
</ul>
```

As part of an HTML page, this would display as:

- The
- Quick
- Brown
- Fox

P.S. A Perl syntactic short cut can be used to construct the list above:

```
out = listToHTML(qw/The Quick Brown Fox/);
```

12. Write a Perl function `hashToHTML()` that returns a string of HTML code that could be used to display a Perl associative array (hash) as an HTML table, e.g.

```
# the hash table ...
colours = ("John"=>"blue", "Anne"=>"red", "Andrew"=>"green");
# and the function call ...
out = hashToHTML(%colours);
```

would result in `$out` having the value ...

```
<table border="1" cellpadding="5">
<tr><th> Key </th><th> Value </th></tr>
<tr><td> Andrew </td><td> green </td></tr>
<tr><td> Anne </td><td> red </td></tr>
<tr><td> John </td><td> blue </td></tr>
</table>
```

As part of an HTML page, this would display as:

Key	**Value**
Andrew	green
Anne	red
John	blue

Note that the hash should be displayed in ascending alphabetical order on key values.

13. Write a perl program that will read in an HTML document and output a new HTML document that contains a table with two cells (in one row). In the left cell should be a copy of the complete original HTML document inside <PRE> tags so we can see the raw HTML. You will need to replace all "<" characters with the sequence "&lt;" and all ">" characters with the sequence "&gt;", otherwise the browser will think they are HTML tags (and we want to see the tags in the left cell). In the right cell just include the HTML body of the document, so we can see what it will look like when rendered by a browser.

14. Write a Perl program that reads in data about student performance in a Prac Exam and computes the overall result for each student. The program takes a *single command line argument*, which is the name of a file containing space-separated text records of the form:

```
studentID  exerciseID  testsPassed  numWarnings
```

There will be one line in the file for each exercise submitted by a student, so a given student may have one, two or three lines of data.
The output is ordered by student ID and contains a single line for each student, in the format:

```
studentID  totalMark  passOrFail
```

The *totalMark* value is computed as follows:

- if an exercise passes all 5 tests, it is awarded a mark of 10 and is *correct*
- if an exercise passes less than 5 tests, it is awarded a mark of *testsPassed/2* and is *incorrect*
- if there are *any* warnings on an exercise, the mark is reduced by 2
- the minimum mark for a given exercise is zero
- the *totalMark* is the sum of the marks for the individual exercises

The *totalMark* value should be display using the `printf` format `"%4.1f"`. A student is awarded a `PASS` if they have 2 or 3 *correct* exercises and is awarded a `FAIL` otherwise. Note that warnings do not cause an exercise to be treated as incorrect.

Sample Marks File	**Corresponding Output**
Command line argument: `marks1`	

```
2121211 ex1 5 0
2121211 ex2 5 0
2121211 ex3 5 0
2233455 ex1 5 0
2233455 ex2 5 1
2233455 ex3 0 1
2277688 ex1 4 0
2277688 ex2 3 0
2277688 ex3 2 1
2277689 ex1 5 0
2277689 ex2 5 0
2277689 ex3 1 1
```

```
2121211 30.0 PASS
2233455 18.0 PASS
2277688  3.5 FAIL
2277689 20.0 PASS
```

15. What does this Perl print and why?

```
@a = (1..5);
@b = grep { $_ = $_ - 3; $_ > 0} @a;
print "@a\n";
print "@b\n";
```

16. What does this Perl print?

```
@vec = map { $_ ** 2 } (1,2,3,4,5);
print "@vec\n";
```