Week 08 Tutorial Sample Answers

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 how often each is used.

Don't count closing tags.

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

For example:

```
$ ./tags.pl https://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 change almost daily.

Sample solution for tags.pl

```
#!/usr/bin/perl -w
# written by andrewt@cse.unsw.edu.au as a COMP2041 example
# fetch specified web page and count the HTML tags in them

# There are better ways to fetch web pages (e.g. HTTP::Request::Common)
# The regex code below doesn't handle a number of cases.  It is often
# better to use a library to properly parse HTML before processing it.
# But beware illegal HTML is common & often causes problems for parsers.

foreach $url (@ARGV) {
    $webpage = `wget -q -O- '$url'`;
    $webpage =~ tr/A-Z/a-z/;
    $webpage =~ s/<!--.*?-->//g; # remove comments
    @tags = $webpage =~ /<\s*(\w+)/g;
    foreach $tag (@tags) {
        $tag_count{$tag}++;
    }
}
foreach $tag (sort keys %tag_count) {
    print "$tag $tag_count{$tag}\n";
}
```

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

```
$ tags.pl -f https://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
```

Sample solution for tags.pl

```perl
#!/usr/bin/perl -w
# written by andrewt@cse.unsw.edu.au as a COMP2041 example
# fetch specified web page and count the HTML tags in them

# The regex code below doesn't handle a number of cases.  It is often
# better to use a library to properly parse HTML before processing it.
# But beware illegal HTML is common & often causes problems for parsers.

use LWP::Simple;

$sort_by_frequency = 0;
foreach $arg (@ARGV) {
    if ($arg eq "-f") {
        $sort_by_frequency = 1;
    } else {
        push @urls, $arg;
    }
}
foreach $url (@urls) {
    $webpage = get $url;
    $webpage =~ tr/A-Z/a-z/;
    $webpage =~ s/<!--.*?-->//g; # remove comments
    @tags = $webpage =~ /<\s*(\w+)/g;
    foreach $tag (@tags) {
        $tag_count{$tag}++;
    }
}
if ($sort_by_frequency) {
    @sorted_tags = sort {$tag_count{$a} <=> $tag_count{$b} || $a cmp $b} keys %tag_count;
} else {
    @sorted_tags = sort keys %tag_count;
}
print "$_ $tag_count{$_}\n" foreach @sorted_tags;
```

Revision questions

The following questions are primarily intended for revision, either this week or later in session. Your tutor may still choose to cover some of these questions, time permitting.

1. Write a Perl function `print_hash()` that displays the contents of a Perl associative array (hash) in the format below (its the format used by the PHP function `print_r()` e.g. the hash table ...

```
%colours = ("John" => "blue", "Anne" => "red", "Andrew" => "green");
```

and the function call ...

```
print_hash(%colours);
```

should produce the output ...

```
[Andrew] => green
[Anne] => red
[John] => blue
```

Since the function achieves its effect via `print`, it doesn't really need to return any value, but since Perl functions typically return *something*, `print_hash` should return a count of the number of items displayed (i.e. the number of keys in the hash table). Note that the hash should be displayed in ascending alphabetical order on key values.

This gives the function as well as some code to test it out:

```perl
#!/usr/bin/perl -w

sub print_hash {
    my (%tab) = @_;
    my $n = 0;
    foreach $k (sort keys %tab) {
        print "[$k] => $tab{$k}\n";
        $n++;
    }
    return $n;
}

%h = (
    "David" => "green",
    "Phil" => "blue",
    "Andrew" => "red",
    "John" => "blue"
);

$nitems = print_hash(%h);
print "#items = $nitems\n";
```

2. A bigram is two words occuring consecutively in a piece of text. Some pairs of words tend to occur more commonly than others as bigrams, e.g `cold beer` or `programming language`.

   Your task is to write a Perl program `bigrams.pl` which reads a piece of text, and prints the words which occur in the text in sorted order, one per line. Each word should be accompanied by the word which most frequently follows it in the text - if several words occur equally often after the word, any of them can be printed. The number of times the word occurs in the text should be indicated as should the number of times the second word follows it. Case should be ignored. For example given this text:

```
$ ./bigrams.pl
Peter Piper picked a peck of pickled peppers;
A peck of pickled peppers Peter Piper picked;
If Peter Piper picked a peck of pickled peppers,
Where's the peck of pickled peppers Peter Piper picked?
Ctrl-D
a(3) peck(3)
if(1) peter(1)
of(4) pickled(4)
peck(4) of(4)
peppers(4) peter(2)
peter(4) piper(4)
picked(3) a(2)
pickled(4) peppers(4)
piper(4) picked(4)
s(1) the(1)
the(1) peck(1)
where(1) s(1)
```

Some notes on the Perl solution below:

- `\W` is a special Perl regexp class which matches any non-word character
- `tr/abc/def/` behaves like the Unix `tr` command
- neither of `tr`'s args is a regexp; but it supports `A-Z` style ranges

- neither of `tr`'s args is a regexp; but it supports `A-Z`-style ranges
- `bigram_count` is a hash where each key is a string and each value is a (reference to a) hash

```perl
#!/usr/bin/perl -w

while ($line = <>) {
    foreach $word (split(/\W+/, $line)) {
        $word =~ tr/A-Z/a-z/;
        $bigram_count{$last_word}{$word}++ if $last_word;
        $last_word = $word;
    }
}
foreach $first_word (sort keys %bigram_count) {
    my $most_common_second_word = "";
    my $most_common_count = 0;
    my $total_count = 0;
    foreach $second_word (sort keys %{$bigram_count{$first_word}}) {
        my $b = $bigram_count{$first_word}{$second_word};
        $total_count += $b;
        if ($b > $most_common_count) {
            $most_common_second_word = $second_word;
            $most_common_count = $b;
        }
    }
    print "$first_word($total_count) $most_common_second_word($most_common_count)\n";
}
```

3. 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
```

Sample Perl solution

```perl
#!/usr/bin/perl -w
die "Usage $0 <n> <m> <column-width>" if @ARGV != 3;
$n = $ARGV[0];
$m = $ARGV[1];
$width = $ARGV[2];
$format = "%${width}d";
foreach $x (1..$n) {
    printf $format, $x;
    foreach $y (1..$m) {
        printf "%${width}d", $x*$y;
    }
    print "\n";
}
```

Sample Python solution

```python
#!/usr/bin/python
import glob, sys, re

if len(sys.argv) != 4:
    sys.stdout.write("Usage: %s <n> <m> <column-width>\n\n" % sys.argv[0])
    sys.exit(1)

n = int(sys.argv[1])
m = int(sys.argv[2])
width = int(sys.argv[3])
format = "%%%dd" % width

for x in range(1, n + 1):
    print(format % x)
    for y in range(1, m + 1):
        sys.stdout.write(' ' + (format % (x * y)))
    print()
```

4. 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.

Perl sample solution

```perl
#!/usr/bin/perl -w
# delete blank lines from specified files

die "Usage: $0 <files>\n" if !@ARGV;

foreach $file (@ARGV) {
    open my $in, '<', $file or die "Can not open $file: $!";
    @lines = <$in>;  # reads entire file into array
    close $in;
    open my $out, '>', $file or die "Can not open $file: $!";
    foreach $line (@lines) {
        print $out $line if $line !~ /^\s*$/;
    }
    close $out;
}
```

Perl sample solution using -i switch

```perl
#!/usr/bin/perl -w -i
while (<>) {
    print if !/^\s*$/;
}
```

Perl sample solution using -i and -p switch

```perl
#!/usr/bin/perl -w -i -p
s/^\s*$//
```

Or from the command line:

```
$ perl -ip -e 's/^\s*$//' file1 file2 file3
```

Python sample solution - based on Perl

```python
#!/usr/bin/python
# delete blank lines from specified files
# simple code which could lose data, if a write error occurs
import sys, re

for filename in sys.argv[1:]:
    with open(filename) as f:
        lines = f.readlines()
    with open(filename,'w') as f:
        for line in lines:
            if not re.match(r'^\s*$', line):
                f.write(line)
```

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

```
$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/);
```

Sample solution for listToHTML

```perl
#!/usr/bin/perl -w

sub listToHTML(@) {
    my (@list) = @_;
    return "" if !@list;
    return "<ul>\n<li>".join("\n<li>", @list)."\n</ul>\n";
}
print listToHTML(@ARGV);
```

6. 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.

This gives the function as well as some code to test it out:

```perl
#!/usr/bin/perl -w

sub hashToHTML {
    my (%tab) = @_;
    my $html = "";

    $html = "<table border=\"1\" cellpadding=\"5\">\n".
        "<tr><th> Key </th><th> Value </th></tr>\n";

    foreach $k (sort keys %tab) {
        $html .= "<tr><td> $k </td><td> $tab{$k} </td></tr>\n";
    }
    $html .= "</table>\n";
    return $html;
}

%h = ("David"=>"green", "Phil"=>"blue", "Andrew"=>"red", "John"=>"blue");

print hashToHTML(%h);
exit;
```

7. Write a Perl program that will read in a 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 "<" and all ">" characters with the sequence ">", 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.

Sample solution for show_html.pl

```perl
#!/usr/bin/perl -w
# inspired by from www.cs.www.cs.rpi.edu/~hollingd/eiw.old/5-Perl/ex6.html

my $html_source = join "", <>;
my $modified_html = $html_source;
$modified_html =~ s/<\s*HEAD[^>]*>.*?<\s*\/HEAD[^>]*>//si;
$modified_html =~ s/<\s*\/?\s*(BODY|HTML)[^>]*>//gsi;

my ($title) = ($html_source =~ /.*<\s*TITLE[^>]*>(.*?)<\s*\/TITLE[^>]*>/si);
$title = "No title" if !defined $title;

$html_source =~ s/</\&lt;/g;
$html_source =~ s/>/\&gt;/g;

print <<eof;
<HTML>
<HEAD>
<TITLE>$title</TITLE>
</HEAD>
<BODY>
<H3 ALIGN=CENTER>HTML-VIEW of $title</H3>
<TABLE BORDER=1 BGCOLOR=WHEAT>
<TR><TD><PRE><FONT SIZE=SMALL>$html_source</FONT></PRE></TD><TD>$modified_html</TD></TR>
</TABLE>
</BODY>
</HTML>
eof
```

8. Write a Perl program that reads in data about student performance in a Prac Exam consisting of 3 exercises 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 should be ordered by student ID and should contain 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`

Sample Perl solution

```perl
#!/usr/local/bin/perl
#
# Prac Exam Exercise
# Author: John Shepherd (sample solution)
#

while (<>) {
    chomp;
    my ($sid,$ex,$tests,$warns) = split;
    if ($tests == 5) {
        $mark = 10;
        $ncorrect{"$sid"}++;
    }
    else {
        $mark = $tests/2.0;
    }
    $mark -= 2 if ($warns > 0);
    $mark = 0 if ($mark < 0);
    $total{$sid} += $mark;
}

foreach $sid (sort keys %total) {
    if ($ncorrect{$sid} >= 2) {
        $passfail = "PASS";
    } else {
        $passfail = "FAIL";
    }
    printf "%s %4.1f %s\n", $sid, $total{$sid}, $passfail;
}
```

Sample Python solution

```python
#!/usr/bin/python
import fileinput, re, sys, collections
ncorrect = collections.defaultdict(int)
total = collections.defaultdict(float)
for line in fileinput.input():
    (sid,ex,tests,warns) = line.split()
    if tests == '5':
        mark = 10
        ncorrect[sid] += 1
    else:
        mark = int(tests)/2.0
    if int(warns) > 0:
        mark = max(0, mark - 2)
    total[sid] += mark
for sid in sorted(total.keys()):
    if ncorrect[sid] >= 2:
        passfail = "PASS"
    else:
        passfail = "FAIL"
    print("%s %4.1f %s" % (sid, total[sid], passfail))
```

9. What does this Perl print and why?

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

It prints:

```
-2 -1 0 1 2
1 2
```

The `grep` function aliases $_ to each list element in turn an executes the code in the block. It returns a list ofthe element for which the last expressione evaluated is true.

`{ $_ = $_ - 3 }` subtracts 3 from each element in `@a`. The `$_ > 0` expression selects positive elements.

10. What does this Perl print?

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

It prints:

```
1 4 9 16 25
```

The `map` function applies the code in the block `{ $_ ** 2 }` to each element in the list, and returns a list containing the tranformed values. The `**` operator does exponentiation; and `$_` refers to the "current" element in the list.

11.