Week 07 Tutorial Sample Answers

1. Write a Perl program `word_frequency.pl` which prints a count of all the words found in its input. Your program should ignore case. It should treat any sequence of alphabetic characters ([a-z]) as a word. It should treat any non-alphabetic character as a space. It should print words and their counts sorted in increasing order of frequency in the format shown in this example:

```
$ ./word_frequency.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
1 s
1 the
1 where
1 if
3 a
4 peck
4 peppers
4 piper
4 peter
4 of
4 pickled
4 picked
```

Sample solution for word_frequency.pl

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

while ($line = <>) {
    $line =~ tr/A-Z/a-z/;
    foreach $word ($line =~ /[a-z]+/g) {
        $count{$word}++;
    }
}
@words = keys %count;
@sorted_words = sort {$count{$a} <=> $count{$b}} @words;
foreach $word (@sorted_words) {
    printf "%d %s\n", $count{$word}, $word;
}
```

2. Write a Perl program `missing_words.pl` which given two files as arguments prints, in sorted order, all the words found in file1 but not file2.

You can assume words occur one per line in each file.

Straight-forward sample solution for missing_words.pl

```perl
#!/usr/bin/perl -w
# print words in file 1 but not file 2

die "Usage: $0 <file1> <file2>\n" if @ARGV != 2;

open my $f, '<', $ARGV[0] or die "Can't open $ARGV[0]: $!";
while ($word = <$f>) {
    chomp $word;
    $w{$word} = "added";
}
close $f;

open my $g, '<', $ARGV[1] or die "Can't open $ARGV[1]: $!";
while ($word = <$g>) {
    chomp $word;
    $w{$word} = "deleted";
}
close $g;

foreach $word (sort keys %w) {
    print "$word\n" if $w{$word} ne "deleted";
}
```

Concise but less obvious sample solution for missing_words.pl

```perl
#!/usr/bin/perl -w
# print words in file 1 but not file 2

die "Usage: $0 <file1> <file2>\n" if @ARGV != 2;

open my $f, '<', $ARGV[0] or die "Can't open $ARGV[0]: $!";
$w{$_}++ while <$f>;
close $f;

open my $g, '<', $ARGV[1] or die "Can't open $ARGV[1]: $!";
delete $w{$_} while <$g>;
close $g;

print sort keys %w;
```

3. A citizen science project monitoring whale populations has files containing large numbers of whale observations. Each line in these files contains:
    - the date the observation was made
    - the number of whales in the pod ("pod" is the collective number for a group of whales)
    - the species of whale

    For example:

```
$ cat whale_observations.txt
01/09/18 21 Southern right whale
01/09/18  5 Southern right whale
02/09/18  7 Southern right whale
05/09/18  4 Common dolphin
05/09/18  9 Pygmy right whale
05/09/18  4 Striped dolphin
05/09/18 35 Striped dolphin
05/09/18  4 Blue whale
```

Write a Perl program `./merge_whales.pl` which reads a file of whale observations and prints them to its standard output, merging adjacent counts from the same day of the same species. For example:

```
$ ./merge_whales.pl  whale_observations.txt
01/09/18 26 Southern right whale
02/09/18 7 Southern right whale
05/09/18 4 Common dolphin
05/09/18 9 Pygmy right whale
05/09/18 39 Striped dolphin
05/09/18 4 Blue whale
```

Sample Perl solution

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

for $filename (@ARGV) {
    open my $f, $filename or die "Can not open $filename\n";

    my $current_date = "";
    my $current_count = 0;
    my $current_species = "";

    while ($line = <$f>) {
        if ($line =~ /^(\S+)\s+(\d+)\s+(.+)\s*$/) {
            my $date = $1;
            my $count = $2;
            my $species = $3;

            if ($species eq $current_species && $date eq $current_date) {
                $current_count += $count;
            } else {
                print "$current_date $current_count $current_species\n" if $current_count;
                $current_date = $date;
                $current_count = $count;
                $current_species = $species;
            }

        } else {
            print "Sorry couldn't parse: $line\n";
        }
    }

    print "$current_date $current_count $current_species\n" if $current_count;
}
```

4. Write a Perl program `./whale_last_seen.pl` which reads a file of whale observations in the same format as the last question and prints the species in alphabetical order, with the date they were last seen.

```
$ ./whale_last_seen.pl  whale_observations.txt
Blue whale 05/09/18
Common dolphin 05/09/18
Pygmy right whale 05/09/18
Southern right whale 02/09/18
Striped dolphin 05/09/18
```

You can assume the file is in chronological order, so the last line in the file for a species will be the last date for the species.

Sample Perl solution

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

for $filename (@ARGV) {
    open my $f, $filename or die "Can not open $filename\n";

    while ($line = <$f>) {
        if ($line =~ /^(\S+)\s+\d+\s+(.+)\s*$/) {
            my $date = $1;
            my $species = $2;

            $last_seen{$species} = $date;
        } else {
            print "Sorry couldn't parse: $line\n";
        }
    }

    foreach $species (sort keys %last_seen) {
        print "$species $last_seen{$species}\n";
    }
}
```

5. Write a Perl program, `phone_numbers.pl` which given the URL of a web page fetches it by running *wget* and prints any strings that might be phone numbers in the web page.

   Assume the digits of phone numbers may be separated by zero or more spaces or hyphens ('-') and can contain between 8 and 15 digits inclusive.

You should print the phone numbers one per line with spaces & hyphens removed.

For example

```
$ ./phone_numbers.pl https://www.unsw.edu.au
20151028
11187777
841430912571345
413200225
61293851000
57195873179
```

Sample Perl solution

```
#!/usr/bin/perl -w
# there are perl libraries which provide a  better way to fetch web pages
foreach $url (@ARGV) {
    open my $f, '-|', "wget -q -O- $url" or die;
    while ($line = <$f>) {
        @numbers = split /[^\d\- ]/, $line;
        foreach $number (@numbers) {
            $number =~ s/\D//g;
            print "$number\n" if length $number >= 8 && length $number <= 15;
        }
    }
    close $f;
}
```

Another sample Perl solution for phone_numbers.1.pl

```
#!/usr/bin/perl -w
# there are perl libraries which provide a  better way to fetch web pages
foreach $url (@ARGV) {
    open my $f, '-|', "wget -q -O- $url" or die;
    while ($line = <$f>) {
        @numbers = $line =~ /[\d\- ]+/g;
        foreach $number (@numbers) {
            $number =~ s/\D//g;
            print "$number\n" if length $number >= 8 && length $number <= 15;
        }
    }
    close $f;
}
```

Python solution

```
#!/usr/bin/python
import sys, re, subprocess
# there are python libraries which provide a  better way to fetch web pages
for url in sys.argv[1:]:
    webpage = subprocess.Popen(["wget","-q","-O-",url], stdout=subprocess.PIPE).communicate()[0]
    for number in re.findall(r'[\d \-]+', webpage):
        number = re.sub(r'\D', '', number)
        if len(number) >= 8 and len(number) <= 15:
            print(number)
```

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. What does each of the following Perl code fragments print (no, don't just clip the lines and pass them to Perl, think about what they're doing):

    a.
    ```
    $x = 'x';
    for ($i = 1; $i <= 3; $i++) {
        $x = "($x)";
    }
    print "$x\n";
    ```

    It iterates three times through the loop, and each iteration wraps a pair of parentheses around what was there on the last iteration so it prints:

    ```
    (((x)))
    ```

b.
```
@hi = split //,"hello";
for ($i = 0; $i < 4; $i++) {
    print $hi[$i];
}
print "\n";
```

The idiom `split //` splits a string into an array of individual characters (in this case `("h","e","l","l","o")`. The loop iterates over the first four of these characters. So it prints:

```
hell
```

c.
```
@vec = (10 .. 20);
print "@vec[1..3]\n";
```

The first statement produces an array containing the integers between 10 and 20 inclusive; the expression in the `print` statement takes a slice of this array from the 2nd to the 4th elements (remember that index values start at zero) so i prints:

```
11 12 13
```

d.
```
foreach $n (1..10) {
    last if ($n > 5);
    print "$n ";
    next if ($n % 2 == 0);
    print "$n ";
}
print "\n";
```

The loop iterates with `$n` set to the values from 1 to 10 inclusive. The `last` terminates the loop as soon as the value of `$n` exceeds five (cf. `break` in C or Java). The `next` starts the next iteration straight away whenever the test succeeds; which occurs for each even number, so that the evens only get printed once so it prints:

```
1 1 2 3 3 4 5 5
```