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

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. Write a Perl program that given the road distances between a number of towns (on standard input) calculates the shortest journey between two towns specified as arguments. Here is an example of how your program should behave.

```
$ ./shortest_path.pl Parkes Gilgandra
Bourke Broken-Hill   217
Bourke Dubbo         23
Bourke Gilgandra      62
Bourke Parkes        71
Canowindra Dubbo      35
Canowindra Gilgandra  13
Canowindra Parkes    112
Dubbo Gilgandra       91
Dubbo Parkes         57
Ctrl-D
Shortest route is length = 105: Parkes Dubbo Canowindra Gilgandra.
```

Fairly obvious Perl sample solution

```perl
#!/usr/bin/perl -w
# find shortest path between two towns

die "Usage: $0 <start> <finish>\n" if @ARGV != 2;
$start = $ARGV[0];
$finish = $ARGV[1];

while (<STDIN>) {
    /(\S+)\s+(\S+)\s+(\d+)/ || next;
    $distance{$1}{$2} = $3;
    $distance{$2}{$1} = $3;
}

$shortest_journey{$start} = 0;
$route{$start} = "";
@unprocessed_towns = keys %distance;
$current_town = $start;
while ($current_town && $current_town ne $finish) {
    @unprocessed_towns = grep {$_ ne $current_town} @unprocessed_towns;

    foreach  $town (@unprocessed_towns) {
        if (defined $distance{$current_town}{$town}) {
            my $d = $shortest_journey{$current_town} + $distance{$current_town}{$town};
            if (!defined $shortest_journey{$town} || $shortest_journey{$town} > $d) {
                $shortest_journey{$town} = $d;
                $route{$town} = "$route{$current_town} $current_town";
            }
        }
    }

    my $min_distance = 1e99;   # must be larger than any possible distance
    $current_town = "";
    foreach $town (@unprocessed_towns) {
        if (defined $shortest_journey{$town} && $shortest_journey{$town} < $min_distance) {
            $min_distance = $shortest_journey{$town};
            $current_town = $town;
        }
    }
}

if (!defined $shortest_journey{$finish}) {
    print "No route from $start to $finish.\n";
} else {
    print "Shortest route is length = $shortest_journey{$finish}:$route{$finish} $finish.\n";
}
```

More concise Perl solution

```
#!/usr/bin/perl -w
# find shortest path between two towns

die "Usage: $0 <start> <finish>\n" if @ARGV != 2;
$start = $ARGV[0];
$finish = $ARGV[1];

while (<STDIN>) {
    /(\S+)\s+(\S+)\s+(\d+)/ || next;
    $distance{$1}{$2} = $3;
    $distance{$2}{$1} = $3;
}

$shortest_journey{$start} = 0;
$route{$start} = "";
$current_town = $start;
while ($current_town && $current_town ne $finish) {
    foreach  $town (keys %{$distance{$current_town}}) {
        my $d = $shortest_journey{$current_town} + $distance{$current_town}{$town};
        next if defined $shortest_journey{$town} && $shortest_journey{$town} < $d;
        $shortest_journey{$town} = $d;
        $route{$town} = "$route{$current_town} $current_town";
    }
    delete $distance{$current_town};
    my $min_distance = 1e99;   # must be larger than any possible distance
    $current_town = "";
    foreach $town (keys %distance) {
        next if !defined $shortest_journey{$town};
        next if $shortest_journey{$town} > $min_distance;
        $min_distance = $shortest_journey{$town};
        $current_town = $town;
    }
}

if (!defined $shortest_journey{$finish}) {
    print "No route from $start to $finish.\n";
} else {
    print "Shortest route is length = $shortest_journey{$finish}:$route{$finish} $finish.\n";
}
```

4. 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:

```
#!/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";
```

5. 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
   - `bigram_count` is a hash where each key is a string and each value is a (reference to a) hash

```
#!/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";
}
```

Revision questions

The remaining tutorial questions are primarily intended for revision - either this week or later in session.
Your tutor may still choose to cover some of the questions time permitting.

COMP[29]041 18s2: 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