Week 09 Weekly Test Sample Answers

Test Conditions

These questions must be completed under self-administered exam-like conditions. You must time the test yourself and ensure you comply with the conditions below.

- You may complete this test in CSE labs or elsewhere using your own machine.
- You may complete this test at any time before **Thursday 06 August 21:00**.
- Weekly tests are designed to act like a past paper - to give you an idea of how well you are progressing in the course, and what you need to work on. Many of the questions in weekly tests are from past final exams.
- Once the first hour has finished, you must submit all questions you've worked on.
- You should then take note of how far you got, which parts you didn't understand.
- You may choose then to keep working and submit test question anytime up to Thursday 06 August 21:00
- However the maximum mark for any question you submit after the first hour will be 50%

You may access this **language documentation** while attempting this test:

- Shell/Regex/Perl quick reference
- full Perl documentation

You may also access manual entries (the `man` command).

Any violation of the test conditions will results in a mark of zero for the entire weekly test component.

Set up for the test by creating a new directory called `test09`, changing to this directory, and fetching the provided code by running these commands:

```
$ mkdir test09
$ cd test09
$ 2041 fetch test09
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

WEEKLY TEST QUESTION:

N Distinct lines

Write a Perl program **distinct_lines.pl** which given a single argument **n** reads lines from standard input until **n** different lines have been read.

It should then print a message (exactly as below) indicating how many lines were read. It should then stop. It should not read futher input

If end-of-input is reached before **n** different lines it should print a message indicating how many lines were read.

Your program should ignore case and white-space when comapring lines.

You can assume your program is given a single positive integer as argument.

```
$ ./distinct_lines.pl 3
hi
hello world
hi
hello world
hello world
bye

3 distinct lines seen after 6 lines read.
```

```
$ ./distinct_lines.pl 3
hi
hello world
   hi
hello        world
   HELLO  world
bye

3 distinct lines seen after 6 lines read.
```

```
$ ./distinct_lines.pl 4
how
are
you
are
how
are
well

4 distinct lines seen after 7 lines read.
```

```
$ ./distinct_lines.pl 3
how
are
you

3 distinct lines seen after 3 lines read.
```

```
$ ./distinct_lines.pl 7
hello
how
are
you
Ctrl-D
End of input reached after 4 lines read - 7 different lines not seen.
```

No error checking is necessary.

Obvious (readable) Perl solution for distinct_lines.pl

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

$n_distinct_lines_needed = shift @ARGV or die;

$n_lines = 0;
while ($line = <STDIN>) {
    $n_lines++;
    $line = lc $line;
    $line =~ s/\s//g;
    $lines_seen{$line}++;
    if (keys %lines_seen >= $n_distinct_lines_needed) {
        print "$n_distinct_lines_needed distinct lines seen after $n_lines lines read.\n";
        exit 0;
    }
}
print "End of input reached after $n_lines lines read - $n_distinct_lines_needed different lines not seen.\n";
```

More concise Perl solution for distinct_lines.pl

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

while (<STDIN>) {
    s/\s//g;
    $lines_seen{lc $_}++;
    if (keys %lines_seen >= $ARGV[0]) {
        print "$ARGV[0] distinct lines seen after $. lines read.\n";
        exit 0;
    }
}

print "End of input reached after $. lines read - $ARGV[0] different lines not seen.\n";
```

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

```
$ 2041 autotest distinct_lines
```

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

```
$ give cs2041 test09_distinct_lines distinct_lines.pl
```

Sample solution for `distinct_lines.pl`

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

$n_distinct_lines_needed = shift @ARGV or die;

$n_lines = 0;
while ($line = <STDIN>) {
    $n_lines++;
    $line = lc $line;
    $line =~ s/\s//g;
    $lines_seen{$line}++;
    if (keys %lines_seen >= $n_distinct_lines_needed) {
        print "$n_distinct_lines_needed distinct lines seen after $n_lines lines read.\n";
        exit 0;
    }
}
print "End of input reached after $n_lines lines read - $n_distinct_lines_needed different lines not seen.\n";
```

Alternative solution for `distinct_lines.pl`

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

$n_distinct_lines_needed = shift @ARGV or die;

$n_lines = 0;
while ($line = <STDIN>) {
    $n_lines++;
    $line = lc $line;
    $line =~ s/\s//g;
    $lines_seen{$line}++;
    if (keys %lines_seen >= $n_distinct_lines_needed) {
        print "$n_distinct_lines_needed distinct lines seen after $n_lines lines read.\n";
        exit 0;
    }
}
print "End of input reached after $n_lines lines read - $n_distinct_lines_needed different lines not seen.\n";
```

Alternative solution for `distinct_lines.pl`

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

while (<STDIN>) {
    s/\s//g;
    $lines_seen{lc $_}++;
    if (keys %lines_seen >= $ARGV[0]) {
        print "$ARGV[0] distinct lines seen after $. lines read.\n";
        exit 0;
    }
}

print "End of input reached after $. lines read - $ARGV[0] different lines not seen.\n";
```

WEEKLY TEST QUESTION:

What have We been Eating

Your task is to write a Shell script **eating.sh** which prints a list of the food we have bought at the supermarket in the last week.

It will be given as its first argument a file containing a list of supermarket bills.

The file will be in this format:

```
[
  [
     {"name": "Rice", "price": "$4.29"},
     {"name": "Butter", "price": "$4.00"}
  ],
  [
     {"name": "Cheese", "price": "$8.82"},
     {"name": "Rice", "price": "$6.84"},
     {"name": "Pasta", "price": "$7.87"}
  ],
  [
     {"name": "Peanut Butter", "price": "$6.93"},
     {"name": "Bread", "price": "$3.32"},
     {"name": "Noodles", "price": "$4.71"}
  ],
  [
     {"name": "Cheese", "price": "$4.60"},
     {"name": "Rice", "price": "$8.23"},
     {"name": "Bread", "price": "$5.99"},
     {"name": "Pasta", "price": "$0.54"},
     {"name": "Ramen", "price": "$0.98"}
  ],
  [
     {"name": "Noodles", "price": "$0.38"}
  ]
]
```

If **eating.sh** was given this file as its first argument it should print

```
Bread
Butter
Cheese
Noodles
Pasta
Peanut Butter
Ramen
Rice
```

bills.json contains an example list of bills

Download bills.json, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/20T2/activities/eating/bills.json .
```

Here is how **eating.sh** should behave:

```
$ eating.sh bills.json
Bread
Butter
Cheese
Noodles
Pasta
Peanut Butter
Ramen
Rice
```

You must print the food in alphabetic order.

Each type of food must be printed only once.

Your function does not have to handle differences in case or white-space.

You can assume the formating of the file is the same as the example above. Your program does not have to handle the many other ways a JSON file might be formatted.

You can assume the "name" and "price" fields are always in the same order as the above example.

Your answer must be Shell. You can not use other languages such as Perl, Javascript, Python or C.

No error checking is necessary.

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

```
$ 2041 autotest eating
```

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

```
$ give cs2041 test09_eating eating.sh
```

Sample solution for `eating.sh`

```
#!/bin/sh

# Tally supermarket bills in a file
# andrewt@unsw.edu.au for a COMP[29]041 exercise
# Lines in file have this format
# {"name": "Peanut Butter", "price": "$6.93"},

egrep '"name"' "$1"|
cut -d\" -f4|
sort|
uniq
```

Alternative solution for `eating.sh`

```
#!/bin/sh

# Tally supermarket bills in a file
# andrewt@unsw.edu.au for a COMP[29]041 exercise
# Lines in file have this format
# {"name": "Peanut Butter", "price": "$6.93"},

egrep '"name"' "$1"|
sed '
    s/.*"name": "//
    s/".*//
    '|
sort|
uniq
```

WEEKLY TEST QUESTION:

Counting A Whale Species in Perl

Your task is to write a Perl program **json_count.pl** which takes as its first argument a whale species and its second argument a file containing a list of whale observations, and prints the number of whales of that species in the file.

The file will be in this format:

```
[
    {
        "date": "09/09/18",
        "how_many": 11,
        "species": "Orca"
    },
    {
        "date": "04/11/17",
        "how_many": 41,
        "species": "Long-finned pilot whale"
    },
    {
        "date": "17/03/18",
        "how_many": 30,
        "species": "Southern right whale"
    },
    {
        "date": "17/08/18",
        "how_many": 31,
        "species": "Orca"
    },
]
```

If **json_count.pl** was given "Orca" as its first argument and the above file as its second argument , it should print 42.

For example:

```
$ ./json_count.pl Orca whales.json
117
$ ./json_count.pl 'Striped dolphin' whales.json
19
$ ./json_count.pl 'Southern right whale' whales.json
64
$ ./json_count.pl 'Whale Shark' whales.json
0
```

You can assume the formating of the file is the same as the example above. Your program does not have to handle the many other ways a JSON file might be formatted.

You can assume the "date", "how_many" and "species" fields in whale observations are always in the same order as the above example.

Your answer must be Perl only. You can not use other languages such as Shell, Python or C.

You may not run external programs, e.g. via system or backquotes.

You may use any Perl module installed on CSE systems.

No error checking is necessary.

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

```
$ 2041 autotest json_count
```

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

```
$ give cs2041 test09_json_count json_count.pl
```

Sample solution for `json_count.pl`

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

# Count  observations of a whale species in a file
# andrewt@unsw.edu.au for a COMP[29]041 exercise

die "Usage: $0 <whale species> <file>\n" if @ARGV != 2;

$target_species = $ARGV[0];
$filename = $ARGV[1];

my $n_whales = 0;

open my $f, $filename or die "Can not open $filename\n";

while ($line = <$f>) {
    if ($line =~ /"how_many": (\d+)/i) {
        $how_many = $1;
    } elsif ($line =~ /"species": "(.*)"/i) {
        my $species = $1;
        if ($species eq $target_species) {
            $n_whales += $how_many;
        }
    }
}

close $f;

print "$n_whales\n";
```

Alternative solution for `json_count.pl`

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

#Count  observations of a whale species in  1 or more files
# andrewt@unsw.edu.au for a COMP[29]041 exercise
# terse less readable version

die "Usage: $0 <whale species> <files>\n" if @ARGV != 2;

$target_species = shift @ARGV;
while (<>) {
    $how_many = $1 if /"how_many": (\d+)/i;
    $n_whales += $how_many if /"species": "$target_species"/i;
}
print "$n_whales\n";
```

Submission

When you are finished each exercise 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 **Thursday 06 August 21:00** to complete this test.

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`)

Test Marks

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 or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

The test exercises for each week are worth in total 1 marks.

The best 6 of your 8 test marks for weeks 3-10 will be summed to give you a mark out of 9.

COMP(2041|9044) 20T2: 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