Objectives

In this Lab, you will:

- Become familiar with the environment used for the final exam
- practice coding under exam conditions

Getting Started

The first 10 minutes of the lab is set aside for you to complete the myExperience survey for COMP(2041|9044). Your tutors will leave the room to ensure your answers stay confidential.
This will also leave time for everyone to arrive and be ready to start the exam.

Logging Into the Exam environment

Log out of your regular account after completing the myExperience survey.

Your tutors will log your lab machine into the exam environment,

When all machines are logged into the exam environment tutors will give you further instructions.

At this stage turn off your phone, put it and laptops out of reach.

Practice Exam

You will have 5 minutes reading time after that the exam runs 60 minutes.

Part 1 has to be submitted in the first 5 minutes and during that time you are not permitted to run xterms, shells, editors, etc. You can view online documentation.

You run "Enter Part 1 Answers" from the right mouse button menu to enter the answers to part 1.

In the final exam you will have 30 minutes to complete part 1.

You run "View Part 2 Questions" from the right mouse button menu to see part 2 questions.

Part 2 answers are entered into the file specified by each question and submitted using the "give" command. The questions include submission instructions. The practice exam will be conducted under simulated exam conditions. which means you can't communicate with other people, can not use your phone/tablet/laptop, ...

You tutor can clarify questions and fix any problems with the exam environment.

They can not help you solve the tasks.

Some later lab sessions may have the same questions as yours. Please don't tell people what questions you had. And equally don't try to find out beforehand what questions were used in other labs. You'll get most value, if this practice exam is as real as possible.

Practice Exam Assessment

This work you submit in the exam environment will be automarked and used to calculate your last lab mark.

You can not submit answers outside the exam environment.

You can only submit answers during the hour of the exam.

Assignment 2 Help

There should be time after the exam if you need help with assignment 2.

Question 1

What does this Shell pipeline print?

```
$ cat input
2nd 3rd
4th 5th 6th
7th 8th 9th 10th
11th
$ cat input|sed 's/[a-j].*/ /g'|sort
```

Question 2

What does q2.pl printwhen run as below?

```
$ cat q2.pl
#!/usr/bin/perl

while (<>) {
    chomp;
    @a = split;
    $h{$a[0]} .= $a[1];
}
print $h{"a"}, "\n";
$ cat input
a 6
b 5
c 4
a 3
b 2
c 1
$ ./q2.pl <input
```

Question 3

What does q3.pl printwhen run as below?

```
$ cat q3.pl
#!/usr/bin/perl

@x = <>;
print $#x+1, "\n";
$ cat input
First line of example text
2nd line of example text
Last line of example text
$ ./q3.pl <input
```

Exercise: female_enrollments

We have student enrolment data in this familiar format:

```
$ cat enrollments.txt
COMP1011|3360379|Costner, Kevin                         |3978/1|M
COMP1011|3360582|Neeson, Liam                           |3711/1|M
COMP2920|3860539|Spears, Brittney                       |3978/3|F
COMP1021|3360582|Neeson, Liam                           |3711/1|M
COMP3411|3860538|Klum, Heidi                            |3978/3|F
COMP3141|3383025|Thorpe, Ian                            |3978/3|M
COMP3891|3860544|Klum, Heidi                            |3978/3|F
```

Write a shell pipeline that given input in this format outputs the student numbers of all female students.
The student numbers should be printed one per line and each student number should be printed once. Only the student numbers should be printed.

The student numbers should be printed in sorted (increasing) order.

Place your answer to this question in a file named `female_enrollments.sh`

For example, given the above input your pipeline should output this:

```
$ female_enrollments.sh <enrollments.txt
3860538
3860539
3860544
```

Your answer must be a shell pipeline. You may not use `while`, `for` or other loops. You may not use Perl or JavaScript. You may use the usual Unix filters.
When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest female_enrollments
```

Autotest Results

97% of **29** students who have autotested **female_enrollments.sh** so far, passed the autotest test.

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

```
$ give cs2041 lab10_female_enrollments female_enrollments.sh
```

Sample solution for `female_enrollments.sh`

```
#!/bin/sh

egrep 'F$'|cut -d\| -f2|sort|uniq
```

Exercise: remove_repeats

Write a program which takes 0 or more arguments and prints some of those arguments.

The first occurrence and only the first occurrence of any argument should be printed.

The arguments should be printed on a single line. A single space should separate each argument.

Your program can **NOT** assume the arguments with be in any particular order.

Make your program behave **exactly** as indicated by the examples below.

It must produce **exactly** the same output as below, except you may print an extra space at the end of the line if you wish.

Your program must be either Shell, Perl or Javascript..

Name your program **remove_repeats.sh remove_repeats.pl** or **remove_repeats.js**

In the examples below replace **remove_repeats.pl** with **remove_repeats.sh** or **remove_repeats.js** you attempt this question in Shell, Perl or Javascript. For example:

```
$ remove_repeats.pl

$ remove_repeats.pl bird
bird
$ remove_repeats.pl bird cow fish
bird cow fish
$ remove_repeats.pl echo echo echo
echo
$ remove_repeats.pl bird cow fish bird cow fish bird
bird cow fish
$ remove_repeats.pl how much wood would a woodchuck chuck
how much wood would a woodchuck chuck
$ remove_repeats.pl a a a a b a
a b
$ remove_repeats.pl a b c d c b a
a b c d
$ remove_repeats.pl d c b d c a a d
d c b a
```

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

```
$ 2041 autotest remove_repeats
```

Autotest Results

94% of **31** students who have autotested **remove_repeats.*** so far, passed all autotest tests.

- **100%** passed test *1*
- **94%** passed test *10*
- **94%** passed test *11*
- **94%** passed test *12*
- **100%** passed test *2*
- **97%** passed test *3*
- **97%** passed test *4*
- **94%** passed test *5*
- **94%** passed test *6*
- **94%** passed test *7*
- **94%** passed test *8*
- **94%** passed test *9*

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

```
$ give cs2041 lab10_remove_repeats remove_repeats.*
```

Sample solution for `remove_repeats.pl`

```
#!/usr/bin/perl -w
foreach $arg (@ARGV) {
    next if $seen{$arg};
    print "$arg ";
    $seen{$arg} = 1;
}
print "\n";
```

Alternative solution for `remove_repeats.pl`

```
#!/usr/bin/perl
my %seen;
print join(" ",grep(!$seen{$_}++, @ARGV)), "\n";
```

Sample solution for `remove_repeats.c`

```c
#include <stdio.h>
#include <string.h>

int
main(int argc, char* argv[]) {
    int i, j;
    for (i = 0; i < argc; i++) {
        for (j = i - 1; j > 0; j--) {
            if (!strcmp(argv[i], argv[j]))
                break;
        }
        if (!j)
            printf("%s ", argv[i]);
    }
    printf("\n");
    return 0;
}
```

Sample solution for `remove_repeats.js`

```javascript
var seen  = new Set()
for (var i = 2; i < process.argv.length; i++) {
    var arg = process.argv[i]
    if (!seen.has(arg)) {
        process.stdout.write(arg + " ");
        seen.add(arg);
    }
}
process.stdout.write("\n");
```

Sample solution for `remove_repeats.v1.pl`

```perl
#!/usr/bin/perl
my %seen;
print join(" ",grep(!$seen{$_}++, @ARGV)), "\n";
```

Sample solution for `remove_repeats.py`

```python
#!/usr/bin/python3
import sys
seen = set()
for arg in sys.argv[1:]:
    if arg not in seen:
        print(arg, end=' ')
        seen.add(arg)
print()
```

Exercise: text_round

Write a program that copies its standard input to standard output but maps all numbers to their nearest whole number equivalent. For example, **0.667** would be mapped to **1**, **99.5** would be mapped to **100**, **16.35** would be mapped to **16**, and so on. All other text in the input should be transferred to the output unchanged.
A *number* is defined as a string containing some digit characters with an optional decimal point (**'.'**) followed by zero or more additional digit characters.

For example **0**, **100**, **3.14159**, **1000.0**, **0.999** and **12345.** are all valid numbers.

For example, given this input:

```
I spent $15.50 for 3.3kg of apples yesterday.
Pi is approximately 3.141592653589793
2000 is a leap year, 2001 is not.
```

your program should produce this output:

```
I spent $16 for 3kg of apples yesterday.
Pi is approximately 3
2000 is a leap year, 2001 is not.
```

Your program must be either Shell, Perl or Javascript.
Name your program **text_round.sh text_round.pl** or **text_round.js**.

In the examples below replace **text_round.pl** with **text_round.sh** or **text_round.js** you attempt this question in Shell, Perl or Javascript.

For example:

```
$ cat text_round_input.txt
I spent $15.50 for 3.3kg of apples yesterday.
Pi is approximately 3.141592653589793
2000 is a leap year, 2001 is not.
$ text_round.pl <text_round_input.txt
I spent $16 for 3kg of apples yesterday.
Pi is approximately 3
2000 is a leap year, 2001 is not.
```

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

```
$ 2041 autotest text_round
```

Autotest Results

89% of **27** students who have autotested **text_round.*** so far, passed all autotest tests.

- **89%** passed test *0*
- **93%** passed test *1*
- **93%** passed test *2*
- **93%** passed test *3*
- **93%** passed test *4*
- **93%** passed test *5*
- **93%** passed test *6*

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

```
$ give cs2041 lab10_text_round text_round.*
```

Sample solution for `text_round.v2.pl`

```
#!/usr/bin/perl -wp
s/(\d+\.\d+)/int($&+0.5)/eg;
```

Sample solution for `text_round.pl`

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

while ($line = <>) {
    my @numbers = $line =~ /(\d+\.\d+)/g;
    foreach $number (@numbers) {
        my $rounded_number = int($number + 0.5);
        $line =~ s/\b$number\b/$rounded_number/;
    }
    print $line;
}
```

Alternative solution for `text_round.pl`

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

while ($line = <>) {
    while ($line =~ /(\d+\.\d+)/) {
        my $number = $1;
        $number =~ /^(\d+)/;
        my $rounded_number = $1;
        if ($number =~ /\.[5-9]/) {
            $rounded_number++;
        }
        $line =~ s/$number/$rounded_number/;
    }
    print $line;
}
```

Alternative solution for `text_round.pl`

```
#!/usr/bin/perl -wp
s/(\d+\.\d+)/int($&+0.5)/eg;
```

Sample solution for `text_round.v1.pl`

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

while ($line = <>) {
    while ($line =~ /(\d+\.\d+)/) {
        my $number = $1;
        $number =~ /^(\d+)/;
        my $rounded_number = $1;
        if ($number =~ /\.[5-9]/) {
            $rounded_number++;
        }
        $line =~ s/$number/$rounded_number/;
    }
    print $line;
}
```

Exercise: reference

Lines of the form **#n** (where **n** is an integer value), should be replaced this by the **n**'th line of input.

This transformation only applies to lines which start with a # character, followed by the digits of a positive integer and then the newline character. No other characters appear on such lines.

You may assume:

Lines are numbered starting from 1,

There are no more than 100 lines in the input,

No line is more than 80 characters long,
All **n** values are valid input line numbers.

No **n** values refer to other #**n** lines.

Your program must be either Shell, Perl or Javascript.

Name your program **reference.sh reference.pl** or **reference.js**.

In the examples below replace **reference.pl** with **reference.sh** or **reference.js**. you attempt this question in Shell, Perl or Javascript. p> For example:

```
$ cat reference_input.txt
line A
line B
line C
#7
line D
#2
line E
$ reference.pl <reference_input.txt
line A
line B
line C
line E
line D
line B
line E
```

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

```
$ 2041 autotest reference
```

Autotest Results

96% of **24** students who have autotested **reference.*** so far, passed the autotest test.

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

```
$ give cs2041 lab10_reference reference.*
```

Sample solution for `reference.pl`

```
#!/usr/bin/perl -w
@a = <STDIN>;
foreach $i (0..$#a) {
    if ($a[$i] =~ /^#(\d+)/) {
        $a[$i] = $a[$1 - 1];
    }
}
print @a;
```

Alternative solution for `reference.pl`

```
#!/usr/bin/perl -w
@a = <STDIN>;
/^#(\d+)/ and $_ = $a[$1 - 1] foreach @a;
print @a;
```