Practice Prac exam

Aims

This exercise aims to give you practice in:

- doing the kinds of questions that appear on the COMP2041 Prac Exam
- experiencing the pressures of solving problems without prior preparation and within a time limit.

Your goal is to complete several small programming exercises.
The assessable part of the lab will be conducted under exam-style conditions, which means you can't communicate with other students and the tutor can only clarify requirements, fix any setup problems, and confirm the tests. They will not help you solve the tasks.

You will use the same exam environment as the final exam and submit your work within the exam environment during the lab.

Your tutor will determine your lab mark for this week based on what you submit from the exam environment during the lab.

Your tutor will setup the exam environment, provide the exam questions and explain more.

Practice Exam: Version 1

Part 1

1. These shell statements are executed:

```
cat numbers|sed 's/[a-z].*[a-z]//g'|sort
```

The file **numbers** contains these 4 lines:

```
2nd 3rd
4th 5th 6th
7th 8th 9th 10th
11th
```

The shell statements execute without errors or warnings. Exactly what output does it produce?

Note: the file **numbers** contains only 4 lines. It contains no blank lines and no leading or trailing white space.

```
11
2
4
7
```

Part 2

2. We have student enrolment data in this familiar format:

```
COMP1011|3360379|Costner, Kevin                      |3978/1|M
COMP1011|3360582|Neeson, Liam                        |3711/1|M
COMP2920|3860538|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.

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

```
3860538
3860544
```

Your answer must be a shell pipeline. You may not use **while**, **for** or other loops. You may not use perl or python. You may use the usual Unix filters.

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

Part 3

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

   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.

   For example, if you program is named `a.out` here is how it should behave :

```
% a.out
% a.out bird
bird
% a.out bird cow fish
bird cow fish
% a.out echo echo echo
echo
% a.out bird cow fish bird cow fish bird
bird cow fish
% a.out how much wood would a woodchuck chuck
how much wood would a woodchuck chuck
% a.out
% a.out a a a a b a
a b
% a.out a b c d c b a
a b c d
% a.out d c b d c a a d
d c b a
```

   Sample Perl solution

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

   One line Perl solution

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

4. 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 maximal string containing one or more digit characters optionally with a decimal point (**'.'**) followed by one or more digit characters. You can assume numbers to do not have extra leading zeros.

   For example **0**, **100**, **3.14159**, **1000.0**, **0.999** and **12345.6789** are all valid numbers.

   For example **007**, **3.**, are not valid numbers.

   1.2.3.4 is not a number but contains the numbers **1.2** and **3.4**

   For example, given this input:

```
I spent $15.50 for 3.3kg of apples yesterday.
Pi is approximately 3.14159265358979
2000 is a leap year, 2001 is not.
1.6.3.4
```

   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.
2.3
```

   Sample Perl solution

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

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

Concise Perl solution

```
#!/usr/bin/perl -w
while (<>) {
    s/(\d+\.\d+)/int($&+0.5)/eg;
    print;
}
```

Different Perl solution

```
#!/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;
}
```

5. Write a program that reads lines of text from its standard input and prints them to its standard output. Except for lines which contain with a '#' character followed by a positive integer.
   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 that: 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, and no n values refer to other #n lines.

Sample Input Data	Corresponding Output
line A line B line C #2 line D #7 line E	line A line B line C line B line D line E line E

Sample Perl solution

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

Terse Perl solution

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

Practice Exam: Version 2

Part 1

1. This Perl code is executed:

```
while (
) {
    chomp;
    @a = split;
    $h{$a[0]} .= $a[1];
}
print "$h{a}\n";
```

with these 6 lines of input:

```
a 6
b 5
c 4
a 3
b 2
c 1
```

The Perl executes without errors or warnings. What output does it produce?

Note: there are 6 lines of input. There are no blank lines and no leading or trailing white space.

```
63
```

Part 2

2. We have student enrolment data in this familiar format:

```
COMP1011|3360379|Costner, Kevin                   |3978/1|M
COMP1011|3364562|Carey, Mary                      |3711/1|M
COMP3311|3383025|Thorpe, Ian                      |3978/3|M
COMP2920|3860448|Steenburgen, Mary                |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 number of all students enrolled in only one course.

The student numbers should be printed one per line and each student's number should be printed once. The student numbers should be printed in sorted order.

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

```
3360379
3360582
3364562
3860448
3860538
3860544
```

Your answer must be a shell pipeline. You may not use **while**, **for** or other loops. You may not use perl or python. You may use the usual Unix filters.

```
cut -d\| -f2|sort|uniq -c|grep '^ *1 '|sed 's/.* //'
```

or

```
cut -d\| -f2|sort|uniq -u
```

Part 3

3. Write a program which takes 0 or more arguments and prints the argumnent that occurs most frequently.
If several arguments occur equally frequently. It should print the first occuring of these.

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

It must produce **exactly** the same output as below.

For example, if you program is named `a.out` here is how it should behave :

```
% a.out
% a.out bird
bird
% a.out bird cow fish
bird
% a.out echo echo echo
echo
% a.out hello echo echo echo hello
echo
% a.out bird cow fish bird cow fish
bird
% a.out 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
5
% a.out a a a a b a
a
% a.out a b c d c b a
a
% a.out d c b d c a a d
d
```

Sample Perl solution

```perl
#!/usr/bin/perl -w
my $max_arg_repetition = 0;
foreach $arg (@ARGV) {
    $arg_count{$arg}++;
    if ($arg_count{$arg} > $max_arg_repetition) {
            $max_arg_repetition = $arg_count{$arg};
    }
}
foreach $arg (@ARGV) {
    if ($arg_count{$arg} == $max_arg_repetition) {
        print "$arg\n";
        last;
    }
}
```

4. Write a program that reads lines containing times in 24-hour format and converts the time to 12-hour (am/pm) form. You can assume input lines contain a single time in the format produced by the standard Unix command `date`.

   Consider the examples given below carefully. They should clarify the format and the conversion required.

   You should match behaviour in the examples below exactly.

   You do not have to handle any other input format.

Sample Input Data

```
Thu Jan  1 00:00:00 UTC 1970
Tue Dec 31 00:19:21 PDT 2011
Sun Apr  9 07:30:24 CHAST 2015
Fri Jun 12 11:19:21 EST 2009
Wed Aug 12 12:00:00 BST 2000
Mon Oct 29 12:49:07 EDT 2009
Sat Nov 30 14:30:24 MST 1999
Sat Sep 30 17:30:24 PDT 2007
Mon Feb 15 23:19:11 WST 2002
Mon Jun 15 23:59:59 MSD 2009
```

Corresponding Output

```
Thu Jan  1 12:00:00am UTC 1970
Tue Dec 31 12:19:21am PDT 2011
Sun Apr  9 07:30:24am CHAST 2015
Fri Jun 12 11:19:21am EST 2009
Wed Aug 12 12:00:00pm BST 2000
Mon Oct 29 12:49:07pm EDT 2009
Sat Nov 30 02:30:24pm MST 1999
Sat Sep 30 05:30:24pm PDT 2007
Mon Feb 15 11:19:11pm WST 2002
Mon Jun 15 11:59:59pm MSD 2009
```

Sample Perl solution

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

while (<>) {
    /\s(\d\d):/ or next;
    my $hour = $1;
    $hour1 = sprintf "%02d", $hour - 12;
    s/$hour:/$hour1:/ if $hour > 12;
    s/00:/12:/ if $hour == 0;
    my $suffix = 'am';
    $suffix = 'pm' if $hour > 11;
    s/:(\d\d) /:$1$suffix /;
    print;
}
```

5. Write a program that copies its standard input to its standard output processing embedded commands as described below.

   - Strings of the form <pathname> should be replaced with the contents of the file named *pathname*.
   - Strings of the form <!command> should be replaced by the output of *command*.

   Multiple embedded commands may appear on a single line.

   You may assume that pathnames do not start with an '!' character.

   You may assume that pathnames and commands do not contain the character '>'.

Sample Input Data

```
Hello, <!echo how> are you
I am in the </etc/timezone>timezone.
```

Corresponding Output

```
Hello, how
 are you
I am in the Australia/Sydney
timezone.
```

```
Lines may have no embedded commands
or several: <!date><!date><!date>
1 empty file's contents: </dev/null>
3 empty files:
</dev/null> </dev/null> </dev/null>
/bin/true prints nothing:
<!/bin/true>
Some commands print <!echo -n no> newlines.
>>> The end. <<<<
```

```
Lines may have no embedded commands
or several: Thu Jun 24 17:21:18 EST 2016
Thu Jun 24 17:21:18 EST 2016
Thu Jun 24 17:21:18 EST 2016
1 empty file's contents:
3 empty files:
/bin/true prints nothing:
Some commands print no newlines.
>>> The end. <<<<
```

Sample Perl solution

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

while ($line = <STDIN>) {
    while ($line =~ s/<!([^>]+)>/<!>/g) {
        $s = `$1`;
        $line =~ s/<!>/$s/;
    }
    while ($line =~ s/<([^>]+)>/<>/g) {
        $s = `cat $1`;
        $line =~ s/<>/$s/;
    }
    print $line;
}
```

Concise Perl solution

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

while (<STDIN>) {
    s/<(!?)([^>]+)>/{$1 eq "!" ? `$2` : `cat $2`}/eg;
    print;
}
```

Practice Exam: Version 3

Part 1

1. This Perl code is executed:

```
while (
```

```
) {
    chomp;
    @a = split;
    $h{$a[0]} .= $a[1];
}
print "$h{a}\n";
```

with these 6 lines of input:

```
a 6
b 5
c 4
a 3
b 2
c 1
```

The Perl executes without errors or warnings. What output does it produce?

Note: there are 6 lines of input. There are no blank lines and no leading or trailing white space.

```
63
```

Part 2

2. We have student enrolment data in this familiar format:

```
COMP1011|3360379|Costner, Kevin                  |3978/1|M
COMP1011|3364562|Carey, Mary                     |3711/1|M
COMP3311|3383025|Thorpe, Ian                     |3978/3|M
COMP2920|3860448|Steenburgen, Mary               |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 number of all students enrolled in only one course.

The student numbers should be printed one per line and each student's number should be printed once. The student numbers should be printed in sorted order.

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

```
3360379
3360582
3364562
3860448
3860538
3860544
```

Your answer must be a shell pipeline. You may not use **while**, **for** or other loops. You may not use perl or python. You may use the usual Unix filters.

```
cut -d\| -f2|sort|uniq -c|grep '^ *1 '|sed 's/.* //'
```

or

```
cut -d\| -f2|sort|uniq -u
```

Part 3

3. Write a program that takes two command-line arguments, a string and a filename.
It should print all the lines in the file that contain the string.

When the lines are printed the occurrences of the string in the line should be surrounded by parentheses.

In other words, the program should behave like a very simple grep (no regexp) but add brackets.

You can assume the file exists and it is readable.

You can assume the string contains only alphanumeric characters, i.e. `[a-zA-Z0-9]` .

For example, if your program was named `a.out` , this is how it should behave:

```
% cat >file
the
quick brown
fox jumps over the
lazy dog
% a.out file e
th(e)
fox jumps ov(e)r th(e)
% cat >poem
How much wood would a woodchuck chuck
if a woodchuck could chuck wood?
% a.out poem wood
How much (wood) would a (wood)chuck chuck
if a (wood)chuck could chuck (wood)?
% a.out poem h
How muc(h) wood would a woodc(h)uck c(h)uck
if a woodc(h)uck could c(h)uck wood?
% a.out poem o
H(o)w much w(o)(o)d w(o)uld a w(o)(o)dchuck chuck
if a w(o)(o)dchuck c(o)uld chuck w(o)(o)d?
% a.out poem How
(How) much wood would a woodchuck chuck
% a.out poem how
%
```

Sample Perl solution

```
#!/usr/bin/perl -w
die "Usage: bgrep <file> <string>" if @ARGV != 2;
open F, "<$ARGV[0]" or die "Can open $ARGV[0]: $!";
while ($line = <F>) {
    # depends on $ARGV[1] not containing regex characters  (see \Q)
    if ($line =~ /$ARGV[1]/) {
        $line =~ s/$ARGV[1]/($&)/g;
        print $line;
    }
}
```

Terse Perl solution

```
#!/usr/bin/perl -w
open F, "<$ARGV[0]" or die;
s/\Q$ARGV[1]/($&)/g && print while <F>;
```

4. We have enrollment data where each line has this format:

```
    CourseCode|StudentID|FamilyName, GivenNames|Program|Gender
```

We wish to convert the data to this format:

```
    CourseCode|StudentID|GivenNames FamilyName|Program|Gender
```

Write a program which reads data in the first format from its standard input and writes it in the second format to its standard output. The following shows an example input/output pair for this program:

Sample Input Data	Corresponding Output
COMP2711|3713452|Ahmad, Warren|3645/2|M COMP1711|3819596|Hernando, Justin Yeong|3979/1|M COMP1001|3953441|Noble, Albert Ka Chuen|4075/3|F COMP1021|3487324|Goolam, Mohammad|3643/2|M COMP9901|3857456|Tinoco, Ling Ling Rachel|2665|F COMP9902|3407207|Rhee, Paul Myung-Won|1650|F COMP4001|3916726|Kota, Tsz Kin|8685/1|M	COMP2711|3713452|Warren Ahmad|3645/2|M COMP1711|3819596|Justin Yeong Hernando|3979/1|M COMP1001|3953441|Albert Ka Chuen Noble|4075/3|F COMP1021|3487324|Mohammad Goolam|3643/2|M COMP9901|3857456|Ling Ling Rachel Tinoco|2665|F COMP9902|3407207|Paul Myung-Won Rhee|1650|F COMP4001|3916726|Tsz Kin Kota|8685/1|M

Sample Perl solution

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

while (<>) {
    chomp;
    @f = split /\|/;
    $f[2] =~ s/(.*), (.*\S)/$2 $1/;
    print join("|", @f), "\n";
}
```

One Line Perl solution

```
#!/usr/local/bin/perl -w
# courtesy aek@cse.unsw.EDU.AU
# split line on '|' and reverse first name and surname in the 3rd field

while (<>) {
        chomp;

        # separate the fields according to the '|' delimiter
        @fields = split /\|/;

        # dump line if not enough fields given
        if ($#fields < 2) { print; next; }

        # process name
        $namefield = $fields[2];
        @nameparts = split ',', $namefield; #seperate surname and first names

        # trim leading and trailing space from names
        $nameparts[0] =~ s/^\s+//;
        $nameparts[0] =~ s/\s+$//;
        $nameparts[1] =~ s/^\s+//;
        $nameparts[1] =~ s/\s+$//;

        # construct new name, and put it back in the array of fields
        $fields[2] = "$nameparts[1] $nameparts[0]";

        # give output.  join fields array with the delimiter
        print join '|', @fields;
        print "\n";
}
```

5. A university has a problem with students accumulating large library fines. The vice-chancellor has decided to expel the student with the largest total library fine. We need a program which tallies the library fines of students and prints the name of the student with the largest total library fine.
   The input to this program will be a series of pairs of student names and library fines.

   The output of this program will be a single student name and a total library fine.

   At this university, students are identified by their given names, which are unique and consist only of lower case alphabetic letters.

   At this university, library fines are always a small positive integer number of dollars.

   Write a program which reads a series of pairs of student names and library fines from its standard input.

   When the end of input is reached your program should print the student with the largest total library fine.

   Your program should place no limit on the number of students and no limit on the number of fines.

   There will be a penalty if your program places any fixed limit on the number of students or the number of fines.

   Make your program behave **exactly** as indicated by the example below. For example:

```
Enter student name: fred
Enter library fine: 3
Enter student name: mary
Enter library fine: 7
Enter student name: fred
Enter library fine: 2
Enter student name: john
Enter library fine: 4
Enter student name: mary
Enter library fine: 1
Enter student name: fred
Enter library fine: 4
Enter student name: <control-d>
Expel fred whose library fines total $9
```

   Note, the `<control-d>` in the example above indicates the user typing this key to indicate the end-of-input. This will not actually appear on the screen. What actually appears when the `<control-d>` key is typed varies between machines.

   Your program does **not** have to print `"<control-d>"` .

   Your program can assume that student names consist only of lower case alphabetic characters, i.e. no space or punctuation characters.

   Your program can assume that student names are at most 32 characters long.

   Your program can assume that fines are always small postive integers.

   Your program can assume that a student's total library fine will fit in an *int*.

   Your program should place no limit on the number of students.

   Your program should place no limit on the number of fines.

   Your program can assume that there is always one student with the largest total library fine (no ties). In other words one student always has a total library fine larger than any other.

   Your program can assume that there is at least one student with a library fine.

   Your program can assume its input is correct.

   No error checking is required.

Sample Perl solution

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

while (1) {
    print "Enter student name: ";
    $name = <STDIN>;
    last if !defined $name;
    chomp $name;
    print "Enter library fine: ";
    $fine = <STDIN> ;
    $total_fine{$name} += $fine;
}

$largest_fine = -1;
$student_to_expel = "";
foreach $student (keys %total_fine) {
    if ($total_fine{$student} > $largest_fine) {
        $largest_fine = $total_fine{$student};
        $student_to_expel = $student;
    }
}
print "Expel $student_to_expel whose library fines total \$$largest_fine\n";
```