Final Exam

Exam Conditions

- You can start reading this exam at **Saturday 22 August 12:50** Sydney time.
- You can start typing at **Saturday 22 August 13:00** Sydney time.
- You have until **Saturday 22 August 16:00** Sydney time to complete this exam
- Only submissions before **Saturday 22 August 16:00** Sydney time will be marked
- You are not permitted to communicate (email, phone, message, talk, ...) with anyone during this exam, except COMP(2041|9044) staff via **cs2041.exam@cse.unsw.edu.au**
- You are not permitted to get help from anyone but COMP(2041|9044) staff during this exam.
- This is a closed book exam.
- You are not permitted to access papers or books.
- You are not permitted to access files on your computer or other computers, except the files for the exam.
- You are not permitted to access web pages or other internet resources, except the web pages for the exam and the online language cheatsheets & documentation linked below
- **Deliberate violation of exam conditions will be referred to Student Integrity as serious misconduct**

Exam Structure

- There are **12** questions on this axam.
- Total mark of questions on this exam is **100**.
- Questions are **NOT** worth equal marks.
- All 12 questions are practical (programming) questions.
- Not all questions may have provided files, You should create any files needed for submission if they are not provided.
- Answer each question in a **SEPARATE** file. Each question specifies the name of the file to use. These are named after the corresponding question number, Make sure you use **EXACTLY** this file name.
- When you finish working on a question submit the files using the **give** command provided in the question. **You may submit your answers as many times as you like.** The last submission **ONLY** will be marked.
- Do not leave it to the deadline to submit your answers. Submit each question when you finish working on it. Running autotests does not automatically submit your code.
- You can verify what submissions you have made with **`2041 classrun -check final_q<N>`**

Language Documentation

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

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

You may also access:

- manual entries via the `man` command
- Texinfo pages via the `info` command
- Perl documentation via the `perldoc` command
- Bash documentation via the `help` command

Special Considerations

> This exam is covered by the Fit-to-Sit policy. That means that by sitting this exam, you are declaring yourself well enough to do so. You will be unable to apply for special consideration after the exam for circumstances affecting you before it began. If you have questions, or you feel unable to complete the exam, contact **cs2041.exam@cse.unsw.edu.au**

If you experience a technical issue before or during the exam, you should follow the following instructions:

Take screenshots of as many of the following as possible:

- error messages
- screen not loading
- timestamped speed tests
- power outage maps
- messages or information from your internet provider regarding the issues experienced

You should then get in touch with course staff via **cs2041.exam@cse.unsw.edu.au** as soon as the issue arises

Getting Started

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

```
$ mkdir -m 700 exam_final
$ cd exam_final
$ 2041 fetch exam_final
```

Or you can download the provided code as a zip file or a tar file.

If you make a mistake and need a new copy of a particular file you can do the follow:

```
$ rm broken-file
$ 2041 fetch exam_final
```

Only files that don't exist will be recreated, all other files will remain untouched

Question 1 (7 MARKS)

We have student enrolment data in this familiar format:

```
$ cat enrollments.txt
COMP1917|3360379|Costner, Kevin Augustus          |3978/1|M
COMP1917|3364562|Carey, Mary                      |3711/1|M
COMP3311|3383025|Thorpe, Ian Augustus            |3978/3|M
COMP2920|3860448|Steenburgen, Mary Nell          |3978/3|F
COMP1927|3360582|Neeson, Liam                    |3711/2|M
COMP3411|3863711|Klum, Heidi June Anne           |3978/3|F
COMP3141|3383025|Thorpe, Ian Augustus            |3978/3|M
COMP3891|3863711|Klum, Heidi June Anne           |3978/3|F
COMP3331|3383025|Thorpe, Ian Augustus            |3978/3|M
COMP2041|3860448|Steenburgen, Mary Nell          |3978/3|F
COMP2041|3360582|Neeson, Liam                    |3711/2|M
COMP3311|3711611|Klum, Mary                      |3978/3|F
COMP3311|3371161|Thorpe, Ian Fredrick            |3711/3|M
COMP3331|5122456|Wang, Wei                       |3978/2|M
COMP3331|5456732|Wang, Wei                       |3648/3|M
COMP4920|5456732|Wang, Wei                       |3648/3|M
```

Note the input is unordered i.e. not sorted in anyway.

You should find a copy of the above data in the file `enrollments.txt`.

Write a shell pipeline that, given student enrollment data in this format, outputs the first name of all students.

The first names should be printed one per line, and each student's first name should be printed once, no matter how many courses they are enrolled in. But, for example, if three students have the same first name, the first name should be printed 3 times.

Only their first names should be printed. The names should be printed in sorted order.

Put your shell pipeline in a file named `./final_q1.sh`

For example, your pipeline should output this:

```
$ ./final_q1.sh  <enrollments.txt
Heidi
Ian
Ian
Kevin
Liam
Mary
Mary
Mary
Wei
Wei
$ ./final_q1.sh  <more_enrollments.txt | head
Aaron
Abdullah
Aditya
Adrian
Alvin
Alvin
Bo
Brendan
Brian
Callum
$ ./final_q1.sh  <more_enrollments.txt | tail
Zeyu
Zeyu
Zeyu
Zeyu
Zeyu
Zhihao
Zhihao
Zihao
Zijian
Ziyang
$ ./final_q1.sh <more_enrollments.txt | wc -l
73
```

NOTE:

Your answer must be a single Shell pipeline.

Your pipeline should take input from standard input.

Your shell pipeline should be placed in the file `./final_q1.sh` For example, if your answer to this question is

```
grep Andrew|sed 's/^/Hello/'|sort
```

then `./final_q1.sh` should contain:

```
$ cat final_q1.sh
#!/bin/dash
grep Andrew|sed 's/^/Hello/'|sort
```

You may use the usual Unix filters.

You may not use `while`, `for`, or other loops.

You may not use Perl, C, Python, or any other language.

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q1
```

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

```
$ give cs2041 final_q1 final_q1.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q1
```

Question 2 (7 MARKS)

Question 2 (7 MARKS)

Write a Perl program that performs the same task as the Shell pipeline in the previous question.

In other words, given student enrollment data in the same format as the last question, it outputs the first name of all students.

The first names should be printed one per line and each student's first name should be printed once, no matter how many courses they are enrolled in. But, for example, if three students have the same first name the first name should be printed 3 times.

Only their first names should be printed. The names should be printed in sorted order.

Put your Perl program in a file named `./final_q2.pl`

For example, your Perl should output this:

```
$ ./final_q2.pl  <enrollments.txt
Heidi
Ian
Ian
Kevin
Liam
Mary
Mary
Mary
Wei
Wei
$ ./final_q2.pl  <more_enrollments.txt | head
Aaron
Abdullah
Aditya
Adrian
Alvin
Alvin
Bo
Brendan
Brian
Callum
$ ./final_q2.pl  <more_enrollments.txt | tail
Zeyu
Zeyu
Zeyu
Zeyu
Zeyu
Zhihao
Zhihao
Zihao
Zijian
Ziyang
$ ./final_q2.pl <more_enrollments.txt | wc -l
73
```

NOTE:

Your answer must be Perl only.

You may use any Perl module installed on CSE systems.

You may not use Shell, C, Python, or any other language.

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

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q2
```

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

```
$ give cs2041 final_q2 final_q2.pl
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q2
```

Question 3 (7 MARKS)

Write a Shell pipeline that, given student enrollment data in the same format as the previous questions, outputs the student number of all students enrolled in exactly two courses.

In other words, given student enrollment data in the same format as the last questions, it outputs the student number of all students enrolled in exactly two courses.

Only the student numbers should be printed. Each student number should be printed once. The student numbers should be printed one per line. The student numbers should be printed in sorted order.

Put your shell pipeline in a file named `./final_q3.sh`

For example, your pipeline should output this:

```
$ ./final_q3.sh  <enrollments.txt
3360582
3860448
3863711
5456732
$ ./final_q3.sh  <more_enrollments.txt
5200211
5202676
5204057
5210651
5217609
5221931
5224839
5226749
5231825
5242415
5246710
5252050
5257279
5265537
5271243
5275189
5276793
5277536
5289500
5294823
5296401
$ ./final_q3.sh  <more_enrollments.txt | wc -l
21
```

NOTE:

Your answer must be a single Shell pipeline.

Your pipeline should take input from standard input.

Your shell pipeline should be placed in the file `./final_q3.sh`. For example, if your answer to this question is

```
grep Andrew|sed 's/^/Hello/'|sort
```

then `./final_q3.sh` should contain:

```
$ cat final_q3.sh
#!/bin/dash
grep Andrew|sed 's/^/Hello/'|sort
```

You may use the usual Unix filters.

You may not use `while`, `for`, or other loops.

You may not use Perl, C, Python, or any other language.

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q3
```

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

```
$ give cs2041 final_q3 final_q3.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q3
```

Question 4 (7 MARKS)

Write a Perl program that performs the same task as the Shell pipeline in the previous question.

In other words, given student enrollment data in the same format as the last questions, it outputs the student number of all students enrolled in exactly two courses.

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

Only the student numbers should be printed. Each student number should be printed once. The student numbers should be printed one per line. The student numbers should be printed in sorted order.

Put your Perl program in a file named `./final_q4.pl`.

For example, your Perl should output this:

```
$ ./final_q4.pl  <enrollments.txt
3360582
3860448
3863711
5456732
$ ./final_q4.pl  <more_enrollments.txt
5200211
5202676
5204057
5210651
5217609
5221931
5224839
5226749
5231825
5242415
5246710
5252050
5257279
5265537
5271243
5275189
5276793
5277536
5289500
5294823
5296401
$ ./final_q4.pl  <more_enrollments.txt | wc -l
21
```

> NOTE:
>
> Your answer must be Perl only.
>
> You may use any Perl module installed on CSE systems.
>
> You may not use Shell, C, Python, or any other language.
>
> You may not run external programs, e.g. via `system()` or backquotes ``.
>
> No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q4
```

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

```
$ give cs2041 final_q4 final_q4.pl
```

To verify your submissions for this activity:

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q4
```

Question 5 (9 MARKS)

Write a Perl program that reads lines of text from its standard input and, if a line contains two or more integers, it swaps the first and last integer.

If a line contains less than two integers, it should be printed unchanged.

If a line contains more than two integers, only the first and last should be swapped.

Put your Perl program in a file named `./final_q5.pl`

For example, your Perl should output this:

```
$ cat swap_numbers.txt
And the first 24 shall become 42 last
1 is the loneliest number
Some lines have no numbers
Powers of 10 are: 1 10 100 1000
The answer is 42.
6 * 7 = 42
$ ./final_q5.sh < swap_numbers.txt
And the first 42 shall become 24 last
1 is the loneliest number
Some lines have no numbers
Powers of 1000 are: 1 10 100 10
The answer is 42.
42 * 7 = 6
```

NOTE:

You should treat **any** sequences of digits as positive integers.

You do not need to consider negative integers.

You do not need to consider floating point numbers, exponential or scientific notation.

Lines may contain multiple integers, and may contain any number of any characters between the integers.

Your program may read all lines before producing any output.

Your program may produce output one line at a time

Your answer must be Perl only.

You may use any Perl module installed on CSE systems.

You may not use Shell, C, Python, or any other language.

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

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q5
```

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

```
$ give cs2041 final_q5 final_q5.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q5
```

Question 6 (9 MARKS)

We wish to save disk space by replacing identical copies of files with symbolic links.

Write a POSIX-compatible shell script `final_q6.sh`, which takes 0 or more names of files as arguments, and prints commands to replace some of the files with symbolic links

replace some of the files with symbolic links.

We need to check the commands before executing them, so `final_q6.sh` should print the commands, *not* execute them.

For each file specified as an argument, if the file has identical contents (bytes) to any previous file specified as an argument, `final_q6.sh` should print a *ln(1)* command which would replace the file with a symbolic link to the previous file.

`final_q6.sh` should just print the *ln(1)* command, it should *not* execute the *ln(1)* command.

If none of the files can be replaced by symbolic links, `final_q6.sh` should print a message exactly as shown in the last example below.

Your script must work when executed with `/bin/dash` on a CSE system, and must only use the external programs listed below.

Make your program produce **exactly** the output indicated by the example below.

For example, here is how your program should behave:

```
$ ./final_q6.sh file0.txt file1.txt file2.txt file3.txt file4.txt file5.txt file6.txt file7.txt file8.txt
ln -s file0.txt file3.txt
ln -s file1.txt file6.txt
ln -s file2.txt file5.txt
ln -s file2.txt file7.txt
$ ./final_q6.sh file6.txt file4.txt file1.txt file2.txt file3.txt file5.txt file7.txt file8.txt file0.txt
ln -s file6.txt file1.txt
ln -s file2.txt file5.txt
ln -s file2.txt file7.txt
ln -s file3.txt file0.txt
$ ./final_q6.sh file8.txt file7.txt file6.txt file5.txt file4.txt file3.txt file2.txt file1.txt file0.txt
ln -s file7.txt file5.txt
ln -s file7.txt file2.txt
ln -s file6.txt file1.txt
ln -s file3.txt file0.txt
$ ./final_q6.sh file1.txt file2.txt file3.txt file4.txt file5.txt file6.txt
ln -s file1.txt file6.txt
ln -s file2.txt file5.txt
$ ./final_q6.sh file0.txt file1.txt file2.txt file3.txt file4.txt
ln -s file0.txt file3.txt
$ ./final_q6.sh file3.txt file0.txt
ln -s file0.txt file3.txt
$ ./final_q6.sh file3.txt file0.txt file3.txt file0.txt file3.txt file0.txt file3.txt file0.txt
ln -s file3.txt file0.txt
$ ./final_q6.sh file1.txt file2.txt file3.txt file4.txt
No files can be replaced by symbolic links
$
```

NOTE:

Your program should **not** change any files or create any links; it should just print commands to do so.

Your program can assume any supplied arguments are the names of ordinary files.

Your program can assume filenames do not contain whitespace, and do not contain slashes.

Your program can **not** assume anything about the contents of the files. The files may contain any number of any character and any number of lines.

You **are** permitted to create temporary files.

You are permitted to use these and only these external programs:

`basename`	`dirname`	`grep`	`rev`	`strings`	`true`
`cat`	`echo`	`head`	`rm`	`tac`	`uniq`
`chmod`	`egrep`	`ls`	`rmdir`	`tail`	`wc`
`cmp`	`expr`	`mkdir`	`sed`	`tee`	`xargs`
`cp`	`false`	`mv`	`seq`	`test`	
`cut`	`fgrep`	`printf`	`sort`	`touch`	
`diff`	`find`	`pwd`	`stat`	`tr`	

You are permitted to use any built-in shell features including:

`cd`	`if`	`while`
`exit`	`read`	
`for`	`shift`	

You may not use non-POSIX-compatible Shell.
You are not permitted to use `/bin/bash` or `/bin/sh`.
You can assume anything that works with the version of `/bin/dash` on CSE systems is POSIX compatible.

You may not use Perl, C, Python, or any other language

You may not use Perl, C, Python, or any other language.

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q6
```

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

```
$ give cs2041 final_q6 final_q6.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q6
```

Question 7 (9 MARKS)

Write a POSIX-compatible shell script which takes, as command-line arguments, an unordered list of positive integers from *n* to *m*, with possibly one integer missing.

Your shell script should print the missing integer, if any. The missing integer will not be *n* or *m*. Only one integer, if any, will be missing. No integer will occur twice.

Your shell script's input will only contain digits.

Your shell script will be run with `/bin/dash`.

Your are permitted to use only the external programs listed below.

Put your Shell program in a file named `./final_q7.sh`.

For example, `./final_q7.sh` should output this:

```
$ ./final_q7.sh 39 45 40 44 41 43
42
$ ./final_q7.sh 6 8 9 1 7 2 3 10 11 12 5
4
$ ./final_q7.sh 1006 1004 1003 1007 1008 1009
1005
$ ./final_q7.sh 26 39 42 28 24 27 30 41 34 36 35 31 32 33 40 38 37 25 29
```

NOTE:

You can assume the command-line arguments will only be positive integers.

You can assume there will be at least two integers as command-line arguments.

You can *not* assume that there will always be a missing number.

Your shell script should produce only 1 line of output.

This line of output should contain only zero or one positive integers.

You **are** permitted to create temporary files.

You are permitted to use these and only these external programs:

`basename`	`dirname`	`grep`	`rev`	`strings`	`true`
`cat`	`echo`	`head`	`rm`	`tac`	`uniq`
`chmod`	`egrep`	`ls`	`rmdir`	`tail`	`wc`
`cmp`	`expr`	`mkdir`	`sed`	`tee`	`xargs`
`cp`	`false`	`mv`	`seq`	`test`	
`cut`	`fgrep`	`printf`	`sort`	`touch`	
`diff`	`find`	`pwd`	`stat`	`tr`	

You are permitted to use any built-in shell features including:

`cd`	`if`	`while`
`exit`	`read`	
`for`	`shift`	

You may not use non-POSIX-compatible Shell.
You are not permitted to use `/bin/bash` or `/bin/sh`.
You can assume anything that works with the version of `/bin/dash` on CSE systems is POSIX compatible.

You may not use Perl, C, Python, or any other language.

> No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q7
```

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

```
$ give cs2041 final_q7 final_q7.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q7
```

Question 8 (9 MARKS)

Write a Perl program that reads lines of text from its standard input and prints them to its standard output with the words which are not equi-lettered removed.

A word is equi-lettered if every character in the word occurs exactly *n* times (for some *n*). Case should be ignored when considering whether a word is equi-lettered.

For example: **Gaga** is equi-lettered because '**g**' occurs twice and '**a**' occurs twice.

For example: **gauge** is not equi-lettered because '**g**' occurs twice but '**u**', '**a**', and '**e**' occur once.

Assume that a word is any sequence of non-whitespace characters, so, for example, **tock-tock** is considered a single-word.

You should print the words separated by a single space character.

Put your Perl program in a file named `./final_q8.pl`.

For example, your Perl should output this:

```
$ cat final_q8.txt
I shall be telling this with a sigh
Somewhere ages and ages hence
Two roads diverged in a wood and I
I took the one less traveled by
And that has made all the difference

Equi: duck bulbul Gaga tocktocktock wwwwweeeee
Not-equi: goose baboon bonobo Guage tock-tock wwwwweee
Equi: xerophytic Deeded sestettes zZz teammate horseshoer happenchance
Not-equi: elephant decorator agaga teammates horseshoe
$ ./final_q8  < final_q8.txt
I be this with a sigh
ages and ages
Two roads in a and I
I the one by
And has made the

Equi: duck bulbul Gaga tocktocktock wwwwweeeee
Not-equi:
Equi: xerophytic Deeded sestettes zZz teammate horseshoer happenchance
Not-equi:
$ ./final_q8  < story.txt | head
THE LEAP-FROG

A Flea, a and a Leap-frog once wanted to
could jump and they the whole world,
and who chose to come to the
festival. famous jumpers they, as
would say, when they met in the

"I give my daughter to him who jumps
the King; "for it is not so amusing
$ ./final_q8  < story.txt | tail
The Flea then went into foreign it is said,
he was

The sat on a bank, and
on things; and he said "Yes, a fine
is fine is what care
about." And then he began his peculiar
song, from we have taken this history; and may,
very be it does stand
printed in black and white.
```

NOTE:

Your program may read all lines before producing any output.

Your program may produce output one line at a time.

You are not permitted to use any Perl modules, e.g. via `use` or `require`.

Your answer must be Perl only.

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

You may not use Shell, C, Python, or any other language.

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q8
```

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

```
$ give cs2041 final_q8 final_q8.pl
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q8
```

Question 9 (9 MARKS)

We need to save an entire directory tree as a single file. The directory tree may consist of many directories and files.

Write a Perl program, `final_q9.pl`, which takes one argument, the pathname of a directory, and which prints a POSIX-compatible shell-script which, when run, re-creates all the directories and files in the directory tree.

For example, the script printed might include `mkdir` commands to re-create directories.

`final_q9.pl` may output any sequence of Shell commands which re-creates the directory tree, but see the restrictions on the Shell command allowed below.

Your solution (`final_q9.pl`) must be Perl only. It can not run external programs and it can not use Perl modules.

For example, these commands create a directory tree named `a`:

```
$ mkdir a
$ mkdir a/b
$ mkdir a/b/c
$ echo hello andrew >a/file1
$ echo bye andrew >a/b/file2
$ echo 1 >a/b/c/one
$ echo 2 >a/b/c/two
$ ls -lR a
a:
total 8
drwxr-xr-x 3 z1234567 z1234567 4096 Aug 19 20:38 b
-rw-r--r-- 1 z1234567 z1234567   13 Aug 19 20:38 file1

a/b:
total 8
drwxr-xr-x 2 z1234567 z1234567 4096 Aug 19 20:38 c
-rw-r--r-- 1 z1234567 z1234567   11 Aug 19 20:38 file2

a/b/c:
total 8
-rw-r--r-- 1 z1234567 z1234567 2 Aug 19 20:38 one
-rw-r--r-- 1 z1234567 z1234567 2 Aug 19 20:38 two
$ cat a/file1
hello andrew
$ cat a/b/file2
bye andrew
```

And this example shows `final_q9` saving the contents of the directory tree `a` in a file named `saved.sh`:

```
$ ./final_q9.pl a >saved.sh
$ ls -l saved.sh
-rw-r--r-- 1 z1234567 z1234567 1234 Aug 19 20:38 saved.sh
```

This example shows `saved.sh`, created by `final_q9.pl` in the last example, restoring the contents of the directory tree `a` after it has been removed.

```
$ rm -rf a
$ ls -lR a
ls: cannot access 'a': No such file or directory
$ cat a/file1
cat: a/file1: No such file or directory
$ /bin/dash ./saved.sh
$ ls -lR a
a:
total 8
drwxr-xr-x 3 z1234567 z1234567 4096 Aug 19 20:38 b
-rw-r--r-- 1 z1234567 z1234567   13 Aug 19 20:38 file1

a/b:
total 8
drwxr-xr-x 2 z1234567 z1234567 4096 Aug 19 20:38 c
-rw-r--r-- 1 z1234567 z1234567   11 Aug 19 20:38 file2

a/b/c:
total 8
-rw-r--r-- 1 z1234567 z1234567 2 Aug 19 20:38 one
-rw-r--r-- 1 z1234567 z1234567 2 Aug 19 20:38 two
$ cat a/file1
hello andrew
$ cat a/b/file2
bye andrew
```

This example shows `saved.sh`, created by `final_q9.pl` previously, restoring the contents of the directory tree `a` in a different directory.

```
$ mkdir new_directory
$ cd new_directory
$ /bin/dash ../saved.sh
$ cat a/file1
hello andrew
$ cat a/b/file2
bye andrew
```

WARNING:

Autotest will only be of limited assistance in debugging your program. Do not expect autotest messages to be easy to understand for this problem. You will need to debug your program yourself.

DANGER:

It is easily possible to destroy many files with *rm(1)*. Do not test this on your working files for this exam, unless you have a backup. Do not assume your program will make a good enough backup.

NOTE:

You can assume the directory tree to be saved contains only directories and regular files. You can assume it does not contain links or other special files. You can assume it does not contain sparse files.

You can assume file and directory names contain only alphanumeric characters (`[a-zA-Z0-9]`) and `'.'`. You can assume file and directory names do not start with `'.'`.

You can assume that all files contain only alphanumeric characters `[a-zA-Z0-9]` and space (`' '`) and new-line (`'\n'`).

Your program does not have to save or restore permissions, modification times, or other file metadata.

You can assume files and directories do not already exist when restoring a directory tree.

`final_q9.pl` must be in Perl only.

`final_q9.pl` is not permitted to use any Perl modules, e.g. via `use` or `require`.

`final_q9.pl` may not run external programs, e.g. via `system()` or backquotes `` ``.

Your generated Shell script is permitted to use these and only these external programs:

`basename`	`cmp`	`diff`	`egrep`	`fgrep`	`head`
`cat`	`cp`	`dirname`	`expr`	`find`	`ls`
`chmod`	`cut`	`echo`	`false`	`grep`	`mkdir`

mv	rm	sort	tail	tr	xargs
printf	rmdir	stat	tee	true	
pwd	sed	strings	test	uniq	
rev	seq	tac	touch	wc	

Your generated Shell script is permitted to use any built-in shell features including:

cd	if	while
exit	read	
for	shift	

Your generated Shell script may not use non-POSIX-compatible Shell.
Your generated Shell script is not permitted to use `/bin/bash` or `/bin/sh`.
You can assume anything that works with the version of `/bin/dash` on CSE systems is POSIX compatible.

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q9
```

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

```
$ give cs2041 final_q9 final_q9.pl
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q9
```

Question 10 (9 MARKS)

We have text output from an optical character recognition (OCR) program. Unfortunately, sometimes the OCR program cannot recognise letters, and in this case, it outputs a possible list of letters in brackets.

For example, the OCR program might output `h(ij)tchhik(ace)r`, which indicates the second letter is an 'i' or 'j', and the second last letter is an 'a', 'c' or 'e'.

Your task is to write a Perl program, `final_q10.pl`, which uses a dictionary to determine what words match these possible letters.

`final_q10.pl` will be given, as a command line argument, the name of a file containing the dictionary, and will be given, on standard input, the OCR text as lines of words separated by single spaces. The words will contain only lowercase letters ([a-z]) and brackets.

It should print out the input text, replacing any strings the OCR program could not recognise with dictionary matches.

For example, given the input `h(ij)tchhik(ace)r`, and assuming the dictionary contains the word `hitchhiker`, and no other words that match, your program will print `hitchhiker`.

It is possible that multiple words in the dictionary will match. In this case, your program should output a list of matching words surrounded by braces.

For example, given the input word `(cr)oaste(dr)`, your program should output `{coasted,coaster,roasted,roaster}`, assuming all these words are in the dictionary. Note the words must be in sorted order.

If no words match, your program should print the string unchanged. For example, given the string `c(yz)t`, and assuming `cyt` and `czt` are not in the dictionary, then your program should output `c(yz)t`

Make your program produces output **exactly** as indicated by the example below:

```
$ cat final_q10.txt
tr(au)th (ij)s beaut(yj)
t(ho)(in)s is the w(aeo)(ry) st(eo)(pq) (ik)n(sz)(ij)(da)e
(abc)(mn)(bcd) (mi)(ij)les t(ou)  g(ao) be(ft)ore (ij) (st)(lm)eep
notaword h(jkl)itchhik(ae)r
$ ./final_q10.pl dictionary.txt < final_q10.txt
truth is beauty
{this,tons} is the {war,way,wey} {step,stop} inside
and miles to go before i sleep
notaword h(jkl)itchhik(ae)r
```

NOTE:

Your answer must be Perl only.

Your program may print an single extra space on the end of lines.

You have been given a file named `dictionary.txt` containing almost 40000 words.

You can assume the file specified as the first argument exists and contains only words, one per line and that these words will only contain only the lower case letters ([a-z]).

Your program should take less than 10 seconds on a CSE server per word using this dictionary.

Your program may read all lines before producing any output.

Your program may produce output one line at a time

You may use any Perl module installed on CSE systems.

You may not use Shell, C, Python, or any other language.

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q10
```

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

```
$ give cs2041 final_q10 final_q10.pl
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q10
```

Question 11 (9 MARKS)

Letters are sometimes mapped to the digits of a phone number like this:

```
a b c    →  2
d e f    →  3
g h i    →  4
j k l    →  5
m n o    →  6
p q r s  →  7
t u v    →  8
w x y z  →  9
```

No letters map to the digits 0 and 1.

Three or four letters map to each of the other 8 digits.

This mapping can be used to map a word to a number. For example, the word "lecture" maps to 5328873.

Your task is to write a Perl program `final_q11.pl` which finds words or pairs of words which map to a number. It should take two arguments: a file of words and a number.

`final_q11.pl` should print all words in the file that map to the number.

It should also print all *pairs* of words that, when concatenated together, map to that number. The pair of words should be separated by a single space.

`final_q11.pl` should match the behaviour below exactly, except it may produce the lines of output in any order.

```
$ ./final_q11.pl dictionary.txt 5328873
lecture
$ ./final_q11.pl dictionary.txt 2442
chic
ah ha
$ ./final_q11.pl dictionary.txt 234567
$ ./final_q11.pl dictionary.txt 2345678
beg lost
$ ./final_q11.pl dictionary.txt 4663364
gone dog
gone fog
good dog
good fog
goof dog
goof fog
home dog
home fog
hone dog
hone fog
hood dog
hood fog
hoof dog
hoof fog
homed oh
honed oh
```

NOTE:

You can assume the file specified as the first argument exists and contains only words, one per line and that these words will only contain only the lower case letters ([a-z]).

You can assume the second argument contains only the digits [2-9].

You have been given a file named `dictionary.txt` containing almost 40000 words.

Your program should take less than 10 seconds per word using this dictionary.

Your answer must be in Perl only.

You are not permitted to use any Perl modules, e.g. via `use`.

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

No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q11
```

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

```
$ give cs2041 final_q11 final_q11.pl
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q11
```

Question 12 (9 MARKS)

You have a summer internship with Facebook, and your first job is to restore the UNSW Confessions page from backups.

Fortunately, we have two backups of all posts. Unfortunately, there was a bug in our backup program: when backing up posts, it added characters to them.

All the original characters of the posts are present and unchanged in the backup of the post but our backup program added an unknown number of characters, and the positions in the post where these characters were added is also unknown.

So, we have two copies of each post, both with an unknown number of characters added in an unknown number of places.

We would like to recover the original post from the two backup copies.

Unfortunately, that is impossible: we can't recover the original post if the bug in the backup program happened to add the same character at the same position to both copies of the post.

You must write a program to do the next best thing: given the two backup copies, it should recover the longest possibility for the original post.

Your program will be given 2 filenames as arguments. These two files contain the two backups of a post.

Your program should print the longest possible string that might be the original post.

If there are multiple such strings it should print all of them, one per line, in sorted order.

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

For example:

```
$ cat backup0_post1.txt
cCarfol Wu pls be mi@ne
$ cat backup1_post1.txt
Caarol W u pls bze mine%%
$ ./final_q12.pl backup0_post1.txt backup1_post1.txt
Carol Wu pls be mine
$ cat backup0_post2.txt
1C2l34i5ntfon frogm Civigl Enggg, r u sisinglngle? xxbll
$ cat backup1_post2.txt
wwClineweton frr4eerom Civil4 E#%ng, r u sing5uujle? fsxxgjg
$ ./final_q12.pl backup1_post2.txt backup2_post2.txt
Clinton from Civil Eng, r u single? xx
$ cat backup0_post3.txt
cooo  uld a phyy   ysio studedededent ever gogogogogo trout with someeee111one studying fiiiine tarts?
$ cat backup1_post3.txt
zcxoyuclvdb na mp.hy,>s?i.o; st'udae!n@t# ev$e%r^ g*o &ou(t wit)h so+meone study_ing fi[ne a]rt|s?\
$ ./final_q12.pl backup1_post3.txt backup2_post3.txt
could a physio student ever go out with someone studying fine arts?
```

> NOTE:
>
> Your answer must be Perl only.
>
> Your program should take less than 10 seconds on a CSE server per word using this dictionary.
>
> Brute force approaches that simply try all possibilities will be far too slow, and will receive few marks.
>
> You can assume the original posts contain at most 128 characters
>
> You can assume that no more than 512 additional characters have been added by the backup program.
>
> You are not permitted to use any Perl modules, e.g. via **use** or **require**.
>
> You may not use Shell, C, Python, or any other language.
>
> No error checking is necessary.

When you think your program is working, you can run some simple automated tests:

```
$ 2041 autotest final_q12
```

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

```
$ give cs2041 final_q12 final_q12.pl
```

To verify your submissions for this activity:

```
$ 2041 classrun -check final_q12
```

Submission

When you are finished working on a question, submit your work by running **give**.

You can run **give** multiple times. Only your last submission will be marked.

Don't submit any questions you haven't attempted.

Do not leave it to the deadline to submit your answers. Submit each question when you finish working on it. Running autotests does not automatically submit your code.

You can check if you have made a submission with **`2041 classrun -check final_q<N>`**:

```
$ 2041 classrun -check final_q1
$ 2041 classrun -check final_q2
...
$ 2041 classrun -check final_q12
```

Remember you have until **Saturday 22 August 16:00** Sydney time to complete this exam (not including any extra time provided by ELS conditions).

Do your own testing as well as running **autotest**

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