THE UNIVERSITY OF NEW SOUTH WALES

TERM 2, 2023

COMP(2041|9044): SOFTWARE CONSTRUCTION

23T2 Practice Exam (22T2 Final Exam)

— Term 2, 2023 —

12 questions — 100 marks

10 minutes reading; 3 hours working

Examination Information

Examination Instructions and Conditions

- You can start reading the text of this examination when instructed to do so by your invigilator.
- You can start working on examination questions when instructed to do so by your invigilator.
- You *must* stop working on examination questions immediately when instructed to do so by your invigilator.
- Only submissions made before this time will be marked.

  For students with approved examination extensions from UNSW Equitable Learning Services,
  You should continue working until your extended working time expires.

  Your invigilator will tell you when this time expires.
- You **must not** communicate with any person during the examination
  except for COMP(2041|9044) Course Staff and Exam Invigilators.
- You are not permitted to talk, email, telephone, message , etc. all
  except to COMP(2041|9044) Course Staff and Exam Invigilators.
- You **must not** get help from anyone during this exam,
  except from COMP(2041|9044) Course Staff and Exam Invigilators.
- You **must not** communicate (email, message, post, ...) your exam answers to anyone, even after the exam has finished.

  There are two sessions, so other students may still be taking the exam after you have finished.

  Additionally Some students have extended time to complete the exam.

  And some students may be taking the exam on a different day.
- Communicating your answers to other students, even after the exam, may be academic misconduct.
- You **must** ensure that, during and after the examination, no other person can access your work.
- You **must not** use code-synthesis tools, such as GitHub Copilot, during this exam.
- Your **zPass** should not be disclosed to any other person. If you have disclosed your **zPass**, you should change it immediately.
- This is a **closed-book examination**.
- You are not permitted to access papers, books, or any other written materials.
- You are not permitted to access files on your computer or other computers, except the files provided by the exam.
- You are not permitted to access web pages or other Internet resources, except the web pages provided by the exam, and the online language documentation linked below.

Deliberate violation of exam conditions is academic misconduct,
and will be referred to the UNSW Student Conduct and Integrity Unit.

Examination Structure

- This examination has **12** questions,
  (The real exam will have 8-12 questions)
  worth a total of **100** marks.
  (The real exam will be worth 100 marks)
  Questions are **not** worth equal marks.
- All 12 questions are practical questions.
- Not all questions may provide files. **You should create any files needed for submission** if they are not provided.
- You **must** answer each question in a separate file. Each question specifies the name of the file to use. Make sure you use *exactly* this file name.
- When you finish working on a question, you should submit your files using the *give* command specified in the question. You **should not** wait until the submission deadline to submit your answers. Running autotests **does not** automatically submit your code.
- **You do not receive additional time for submitting your answers.**

  You must submit all questions before the end of the 3 hour exam period.
  Failing to logout *immediately* at the end of the exam period is academic misconduct.
- You may submit as many times as you like; *only* the last submission will be marked.
- You can verify what submissions you have made with `2041 classrun -check`

Available Resources: Language Documentation

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

- manual entries, via the *man* command.
- Texinfo pages, via the *info* command.
- shell help, via the `help` builtin.
- python help, via the `python3 -c 'help()'` command.
- Python quick reference
- Full Python3.9 Documentation

Troubleshooting

If you are having issues working on the exam:

- Immediately inform your invigilator.

Fit-to-Sit

This exam is covered by UNSW's 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.

Getting Started

All provided files for this exam are located in your home directory.

simply open a terminal or text editor in your home directory and you will be able to access all the files.

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

```
$ rm <broken-file>
$ 2041 fetch exam_practice
```

Only files that don't exist will be recreated.
All other files will remain untouched.

Question 1 (10 MARKS)

You have been given the file `awards.psv`

This file contains information about Nobel Prize, Turing Award and Fields Medal winners

Each line in the file contains the following six fields:

1. Award Name
2. Award Year
3. Winner Name
4. Winner Gender
5. Winner Country
6. Winner Birth Year

Each field is separated from the next field by a pipe character (`|`).

Your task is to write four *grep* commands which will print specified lines from this file.

You have been given the file `practice_q1.txt` .

In `practice_q1.txt` add answers to the four questions.

Add each answer in the specified location.

Do not add, remove, or change any other text in the file.

The autotest depends on the exact format of the file.

Each question should be answered with a single *grep* command.

You may use any regular expression syntax supported by *grep*.

You may use any options supported by *grep* except `-P/--perl-regexp` .

1. **Question 1**

   Write a *grep* command that will print the lines in `awards.psv` that contain awards won by an Australian.

2. **Question 2**

   Write a *grep* command that will print the lines in `awards.psv` that contain a female ACM Turing Award winner.

3. **Question 3**

   Write a *grep* command that will print the lines in `awards.psv` that contain Nobel Prizes won by a person born between 1990 and 1999 (inclusive).

4. **Question 4**

   Write a *grep* command that will print the lines in `awards.psv` that contain awards won by a person with first name, last name, and middle initial starting with the same letter.

Your example to each question will be a single grep command.

You can test the command by running it in the terminal.

For example, if you think the answer to question 1 is `grep -E 'Andrew' awards.psv` ,
you can test it:

```
$ grep -E 'Andrew' awards.psv
ACM Turing Award|2000|Andrew Chi-Chih Yao|Male|China|1946
Nobel Prize for medicine|1963|Andrew F. Huxley|Male|United Kingdom|1917
Nobel Prize for medicine|2006|Andrew Z. Fire|Male|United States|1959
Nobel Prize for medicine|1977|Andrew V. Schally|Male|United States|1926
ACM Turing Award|2017|David Andrew Patterson|Male|United States|1947
```

Then add it to `practice_q1.txt` .

Autotest will extract your answers from `practice_q1.txt` and do a simple test.

> NOTE:
>
> Your answer for each question **must** be a single *grep* command only
> For example: `grep -E Andrew awards.psv`
>
> You **may not** use the `-P/--perl-regexp` option to grep
>
> You **may** use any other *grep* option, for example you can use grep's `-w` option.
>
> You **may** quote the arguments to grep.
>
> You **may not** use any commands other than *grep*, for example, you can not use *sed* or cut(1).
>
> You **can not** use pipes, or I/O redirection.
>
> You **can** assume that the pipe character (`|`) will only appear as a field separator.

> Note the **input is unordered**.
> It is not sorted in any way.
>
> You **may not** use `Shell` , `C` , `Python` , `Perl` , or any other language.

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

```
$ 2041 autotest practice_q1
```

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

```
$ give cs2041 practice_q1 practice_q1.txt
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 2 (9 MARKS)

We have student enrolment data in this familiar format:

```
$ cat enrollments.txt
COMP1511|3360379|Costner, Kevin Augustus          |3978/1|M
COMP1511|3364562|Carey, Mary                      |3711/1|M
COMP3311|3383025|Thorpe, Ian Augustus            |3978/3|M
COMP4920|3860448|Steenburgen, Mary Nell          |3978/3|F
COMP1521|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
```

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

Each line in the file contains the following five fields:

1. UNSW Course Code
2. Student ID Number
3. Student Name (surname(s), given-names(s))
4. Student Plan
5. Student Gender

Each field is separated from the next field by a pipe character (`|`).

In **practice_q2.sh**, write a shell pipeline that,
given student enrollment data, on stdin, in the above format,
will output the **surnames** (family names) of **male** students.

Each surname should be printed only once

The surnames should be printed in alphabetical order.

> NOTE:
>
> The input is unordered.
> In other words it is not sorted in any way.
>
> The last field contains **M** if a student identifies as **male**.
>
> A **comma** separates the surname(s) from the given-name(s).

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

```
$ ./practice_q2.sh < enrollments.txt
Carey
Costner
Neeson
Thorpe
Wang
```

Using the additional data file provided,
your pipeline should output this:

```
$ ./practice_q2.sh < more_enrollments.txt
Bai
Bian
Cai
Cui
Guo
Islam
Kumar
Li
Long
Lu
Luong
Luu
Murray
Ngo
Saha
Tran
Trinh
Wu
Xia
Xiao
Yao
Ye
Yip
Yong
Zeng
```

NOTE:

You **may** assume that the data always has 5 fields.

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 `practice_q2.sh`
For example, if your answer to this question is:

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

then `practice_q2.sh` should contain:

```
$ cat practice_q2.sh
#! /bin/dash
grep "Andrew" | sed 's/^/Hello/' | sort
```

You **may** use the standard UNIX filters, including: *cut*, *grep*, *head*, *sed*, *sort*, *tail*, *uniq*, *wc*, Etc.

You **may not** use `if`, `while`, `for`, or other shell syntax.

You **may not** use `&&`, `||`, `;`, or other shell syntax.

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

You **may not** use `awk`.

You **may not** create temporary files.

No error checking is necessary.

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

```
$ 2041 autotest practice_q2
```

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

```
$ give cs2041 practice_q2 practice_q2.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 3 (9 MARKS)

Write a Python program **practice_q3.py** that performs the same task as the Shell pipeline in the previous question.

In other words: given data in the same format as the last question on standard input,
output the surnames (family names) of **male** students.

Each surname should be printed only once

The surnames should be printed in alphabetical order.

For example:

```
$ ./practice_q3.py < enrollments.txt
Carey
Costner
Neeson
Thorpe
Wang
```

> NOTE:
>
> Your program **should** read input from standard input.
>
> Your answer **must** be `Python` only.
>
> You **may not** use `Shell`, `C`, `Perl`, or any other language.
>
> You **may not** run external programs, e.g. via the `subprocess` module, or any other method.
>
> You **may** `import` any other standard `Python` module installed at CSE.
>
> **No** error checking is necessary.

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

```
$ 2041 autotest practice_q3
```

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

```
$ give cs2041 practice_q3 practice_q3.py
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 4 (8 MARKS)

Write a POSIX-compatible shell script `./practice_q4.sh` that takes a single command-line argument, a filename.

The file will contain an unordered list of positive integers, one per line, from *n* to *m*, with possibly one integer missing.

Your shell script should print the missing integer, if there is a missing integer.

If there is no missing integer your script should print nothing.

No integer will occur twice.

At most one integer will be missing.

The missing integer will not be *n* or *m*.

For example, `./practice_q4.sh` should output this:

```
$ cat numbers_1.txt
39
45
40
44
41
43
$ ./practice_q4.sh numbers_1.txt
42
$ cat numbers_2.txt
6
8
9
1
7
2
3
10
11
12
5
$ ./practice_q4.sh numbers_2.txt
4
$ ./practice_q4.sh numbers_3.txt
1005
$ ./practice_q4.sh numbers_4.txt
$
```

NOTE:

You can assume there is always exactly one command-line argument, a valid filename.

You can assume the file contains only positive integers, one per line.

You can assume there will be at least two integers in the file.

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

You can assume that at most one integer integer is missing.

Your script should produce at most 1 line of output.

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

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

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

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

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

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 `awk`.

No error checking is necessary.

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

```
$ 2041 autotest practice_q4
```

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

```
$ give cs2041 practice_q4 practice_q4.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 5 (8 MARKS)

In **question 1** you were given the file `awards.psv`

containing information about Nobel Prizes, Turing Awards and Fields Medal winners

This time we are interested in the years particular awards were not given, between when it was first offered and when it was last offered.

Write a POSIX-compatible Shell script **practice_q5.sh** that prints all years that an award was not given in.

practice_q5.sh will be given two command-line arguments, a regular expression and a file name,
practice_q5.sh should print in sorted order the years that no award whose **entire**name matches the regex given.

The entire name must match the regex not just a substring.

For example if the regex is `Nobel Prize` , then awards for `Nobel Prize for physics` should match.

But if the regex is `Nobel Prize.*` , then awards for `Nobel Prize for physics` should match.

You should not hard code the name of this file in your script, the name of the data file will be given as the second command-line argument.

If there are no awards that match the regex, then **practice_q5.sh** should print an message to stdout as below.

For example, the Nobel prize for physics was first awarded in 1901 and last awarded in 2021.
It was not awarded in these six years:

```
$ ./practice_q5.sh 'Nobel Prize for physics' awards.psv
1916
1931
1934
1940
1941
1942
```

More example output:

```
$ ./practice_q5.sh 'Nobel Prize for (chemistry|physics)' awards.psv
1916
1940
1941
1942
$ ./practice_q5.sh 'Nobel Prize for.*' awards.psv
1940
1941
1942
$ ./practice_q5.sh 'COMP2041 Distinguished Achievers' awards.psv
No award matching 'COMP2041 Distinguished Achievers'
```

NOTE:

You **can** assume that the pipe character (`|`) will only appear as a field separator.

Note the input is unordered. It is not sorted in any way.

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

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

See `man 1 <program>` for more info on any program

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

case	cd	exit

```
for                             read                            while
if                              shift
```

See `man 1 dash` for more info on any built-in

See `help <built-in>` for *bash* info on any built-in (might not be POSIX-compatible)

You **may not** use non-POSIX-compatible shell features.

You **are not permitted** to use `/bin/bash`, `/bin/sh`, or any other shell.

Make the first line of your shell-script `#!/bin/dash`

You **can** assume that anything that works with the version of `/bin/dash` on CSE systems is POSIX compatible.

You **may not** use `Perl`, `C`, `Python`, `awk` or any language other than shell.

You **may not** create temporary files.

No error checking is necessary.

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

```
$ 2041 autotest practice_q5
```

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

```
$ give cs2041 practice_q5 practice_q5.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 6 (8 MARKS)

Two files are **mirrored** if they contain the same lines but in the reverse order to each other.

Write a Python program **practice_q6.py** that given two filenames checks if the two files are **mirrored**.

practice_q6.py should always produce one line of output.

If the two files are **mirrored practice_q6.py** should print a message indicating this.

If the two files are not **mirrored practice_q6.py** should print a message indicating why not.

Match the example output below exactly:

If the two files contain a different number of lines,
practice_q6.py should print a single line indicating this, including the number of lines in each file.

This message should be printed in **exactly** the format below.

```
$ echo hello >file1
$ echo Andrew >>file1
$ echo Dylan >file2
$ cat file1
hello
Andrew
$ cat file2
Dylan
$ ./practice_q6.py file1 file2
Not mirrored: different number of lines: 2 versus 1
$ ./practice_q6.py file2 file1
Not mirrored: different number of lines: 1 versus 2
```

If the files contain the same number of lines but a line in the first file
is not the same as the corresponding line in the second file in reverse order,
practice_q6.py should print a single line indicating this.

The message should indicate the line number in the first file (only).

The message should be printed in **exactly** the format below.

```
$ echo hello >>file2
$ cat file1
hello
Andrew
$ cat file2
Dylan
hello
$ ./practice_q6.py file1 file2
Not mirrored: line 2 different
$ ./practice_q6.py file2 file1
Not mirrored: line 1 different
```

Note, message should only be for the first line which is different.
The example below shows files where multiple lines are different.

```
$ sed –i s/hello/hi/ file1
$ cat file1
hi
Andrew
$ cat file2
Dylan
hello
$ ./practice_q6.py file1 file2
Not mirrored: line 1 different
$ ./practice_q6.py file2 file1
Not mirrored: line 1 different
```

The example below shows **mirrored** files.

```
$ echo hello >file3
$ echo Dylan >>file3
$ cat file3
hello
Dylan
$ cat file2
Dylan
hello
$ ./practice_q6.py file2 file3
Mirrored
$ seq 1 5 >a
$ seq 5 –1 1 >b
$ cat a
1
2
3
4
5
$ cat b
5
4
3
2
1
$ ./practice_q6.py a b
Mirrored
```

NOTE:

Your program should always produce exactly one line of output.

Your program can assume it is given exactly two command-line arguments: the names of two files.

You can assume all lines in the two files are terminated with a single '\n' byte.

You can assume the two files contain only ASCII bytes.

You may read all of both files before producing output.

You can assume the files are small enough to fit in memory, e.g. the lines can be read into a list.

Your answer **must** be `Python` only.

You **may not** use `Shell`, `C`, `Perl`, or any other language.

You **may not** run external programs, e.g. via the `subprocess` module, or any other method.

> You **may** `import` any other standard `Python` module installed at CSE.
>
> **No** error checking is necessary.

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

```
$ 2041 autotest practice_q6
```

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

```
$ give cs2041 practice_q6 practice_q6.py
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 7 (8 MARKS)

We have many executable scripts which we want to rename with an extension if possible.

Reminder, a filename extension is a suffix on the end of a filename that indicates the file contents.
For example, an extension of **.sh** can be used to indicate a file contains a shell script.

Write a POSIX-compatible Shell script `practice_q7.sh` which given one or more filenames as command-line arguments,
prints shell commands to add extensions to the filenames.

We need to check the commands before executing them, so `practice_q7.sh` should **print the commands, not execute them**.

For each filename, `practice_q7.sh` should print one line of output to `stdout`,
matching the example output below **exactly**.

- If the filename already has an extension (contains a '.'),
  print a line indicating this.
- If the file does not start with **#!**,
  print a line indicating this.
- If the **#!** line does not contain any of the strings, **perl**, **python** or **sh**,
  print a line indicating this.
- If a file already exists with the new filename created by adding the appropriate extension,
  print a line indicating this.
- Otherwise, **practice_q7.sh** should print a line containing a command to rename the file.

The line printed for each filename should match the example output below **exactly**.
For example:

```
$ touch script.rs
$ ./practice_q7.sh script.rs
# script.rs already has an extension
$ ./practice_q7.sh script1
# script1 does not have a #! line
$ ./practice_q7.sh script2
# script2 no extension for #! line
$ practice_q7.sh script3
mv script3 script3.py
$ ./practice_q7.sh script*
# script1 does not have a #! line
# script2 no extension for #! line
mv script3 script3.py
mv script4 script4.pl
mv script5 script5.sh
$ touch script4.pl
$ ./practice_q7.sh script4
# script4.pl already exists
```

> NOTE:
>
> A **#!** line must be the first line in a file.
>
> Do not examine other lines in the files.
>
> Do not assume anything about the format of **#!** lines.

If the string **python** appears anywhere in a **#!** line, the appropriate suffix for the file is **.py**

If the string **perl** appears anywhere in a **#!** line, the appropriate suffix for the file is **.pl**

If the string **sh** appears anywhere in a **#!** line, the appropriate suffix for the file is **.sh**

Your program should only add **.sh**, **.pl** and **.py** extensions.
It should not add other extensions.
It should instead print the message shown in the example output.

Your program should not actually rename the files.

It should only print the command to stdout.

Your program can assume the filenames it is given exist.

You can assume filenames contain only letters, digits and the characters '.' and '_'.

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

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

See `man 1 <program>` for more info on any program

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

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

See `man 1 dash` for more info on any built-in

See `help <built-in>` for *bash* info on any built-in (might not be POSIX-compatible)

You **may not** use non-POSIX-compatible shell features.

You **are not permitted** to use `/bin/bash`, `/bin/sh`, or any other shell.

Make the first line of your shell-script `#!/bin/dash`

You **can** assume that anything that works with the version of `/bin/dash` on CSE systems is POSIX compatible.

You **may not** use `Perl`, `C`, `Python`, `awk` or any language other than shell.

You **may not** create temporary files.

No error checking is necessary.

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

```
$ 2041 autotest practice_q7
```

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

```
$ give cs2041 practice_q7 practice_q7.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 8 (8 MARKS)

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

Write a POSIX-compatible shell script `practice_q8.sh`, which takes 0 or more names of files as arguments, and prints commands to replace some of the files with symbolic links.

We need to check the commands before executing them, so `practice_q8.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, `practice_q8.sh` should print a *ln* command which would replace the file with a symbolic link to the previous file.

`practice_q8.sh` should just print the *ln* command, it should *not* execute the *ln* command.

If none of the files can be replaced by symbolic links, `practice_q8.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:

```
$ ./practice_q8.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
$ ./practice_q8.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
$ ./practice_q8.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
$ ./practice_q8.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
$ ./practice_q8.sh file0.txt file1.txt file2.txt file3.txt file4.txt
ln -s file0.txt file3.txt
$ ./practice_q8.sh file3.txt file0.txt
ln -s file3.txt file0.txt
$ ./practice_q8.sh file3.txt file0.txt file3.txt file0.txt file3.txt file0.txt file3.txt file0.txt
ln -s file3.txt file0.txt
$ ./practice_q8.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 use these and only these external programs:

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

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

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

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, awk, or any other language.

You **may not** create temporary files.

No error checking is necessary.

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

```
$ 2041 autotest practice_q8
```

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

```
$ give cs2041 practice_q8 practice_q8.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 9 (8 MARKS)

Write a Python program, `practice_q9.py` , which takes two command line argument, a positive integer **n** and the name of a file.

the single positive integer, **n**, indicating a maximum desired line length.

Your program should change the file in the following way:

- Lines containing **n** characters or less, not including the new line, should not be changed.
- Lines not containing a space character (' ') should not be changed.
- Lines containing more than **n** characters with a space in the first **n** characters,
  should have the last space in the first **n** characters changed to a newline character ('\n').
- Lines containing more than **n** characters without a space in the first **n** characters,
  should have the first space on the line changed to a newline character ('\n').

The above rules should also be applied to new lines as they are created.

Your program should not print anything to stdout.
The only thing it should do is change the file in-place.

For example:

```
$ echo hello there how are you today > hello.txt
$ ./practice_q9.py 80 hello.txt
$ cat hello.txt
hello there how are you today
$ ./practice_q9.py 23 hello.txt
$ cat hello.txt
hello there how are
you today
$ ./practice_q9.py 12 hello.txt
$ cat hello.txt
hello there
how are you
today
$ ./practice_q9.py 6 hello.txt
$ cat hello.txt
hello
there
how
are
you
today
$ cp frost.txt f.txt
$ cat f.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.
$ ./practice_q9.py 20 f.txt
$ cat f.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.
$ ./practice_q9.py 10 f.txt
$ cat f.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.
```

Note `practice_q9.py` printed nothing - it changed the file it was given as argument.

Make sure your program does this.

> NOTE:
>
> Your program can assume it is given one argument which is the name of a file.

> Your program can assume the file exists.
>
> You can assume the files contains only ASCII bytes.
>
> You can assume the file is small enough to fit in memory, e.g. the lines can be read into a list.
>
> Your program should print nothing to stdout.
>
> You **are** permitted to create temporary files.
>
> Your answer **must** be `Python` only.
>
> You **may** `use` any standard `Python` modules.
> Unless they are otherwise disallowed by the following restrictions.
>
> You **may not** use `Shell`, `C`, `Perl`, or any other language.
>
> You **may not** run external programs, e.g. via the `subprocess` module, or any other method.
>
> You **may not** import, or otherwise use, the `textwrap` module.
>
> **No** error checking is necessary.

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

```
$ 2041 autotest practice_q9
```

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

```
$ give cs2041 practice_q9 practice_q9.py
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 10 (8 MARKS)

Write a Python program, `practice_q10.py`, that reads lines of text from its standard input and prints them to its standard output with the words which are not ***balanced*** removed.

A word is ***balanced*** if every character in the word occurs exactly *n* times (for some value of *n*).

Case should be ignored when considering whether a word is balanced.

For example: **Gaga** is balanced because '**g**' occurs twice and '**a**' occurs twice.

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

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

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

Match the example output below **exactly**. For example:

```
$ cat frost.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.
$ ./practice_q10.py < frost.txt
I be this with a sigh
ages and ages
Two roads in a and I --
I the one by,
And has made the
$ cat interesting_words.txt
1 duck   bulbul  Gaga    tocktocktock  wwwwweeeee
2 goose baboon bonobo   Guage tock-tock  wwwwweee
3 xerophytic   Deeded sestettes     zZz teammate horseshoer happenchance
4   elephant decorator    agaga  teammates            horseshoe
$ ./practice_q10.py <  interesting_words.txt
1 duck bulbul Gaga tocktocktock wwwwweeeee
2
3 xerophytic Deeded sestettes zZz teammate horseshoer happenchance
4
$ ./practice_q10.py < 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
$ ./practice_q10.py < 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.

Your program can assume its input contains only ASCII bytes.

Your answer **must** be `Python` only.

You **may** `use` any standard `Python` modules.
Unless they are otherwise disallowed by the following restrictions.

You **may not** use `Shell`, `C`, `Perl`, or any other language.

You **may not** run external programs, e.g. via the `subprocess` module, or any other method.

No error checking is necessary.

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

```
$ 2041 autotest practice_q10
```

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

```
$ give cs2041 practice_q10 practice_q10.py
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 11 (8 MARKS)

We need a program to check if the same files are present in two directory trees.

Write a POSIX-compatible Shell script, `practice_q11.sh`, which is given either 2 arguments which are the pathnames of directories.

`practice_q11.sh` should print a single line of output containing 4 integers:

- number of files that are present at the same position in both directory trees and are the same size.
- number of files that are present at the same position in both directory trees but are different sizes.
- number of files that are present only in the first directory tree.
- number of files that are present only in the second directory tree.

Note, a file needs to be present in both directory trees at the same relative pathname to be counted.

For example, if we are comparing the directory trees **dir1** and **dir2**. and the file **dir1/2041/lab09/answer.py** exists in the first directory tree,

We consider it present in the second directory tree only if the file **dir2/2041/lab09/answer.py** exists.

Other files named **answer.py** elsewhere in the second directory tree, e.g. **dir2/2521/lab05/answer.py** are not counted.

When a file is present at the same relative pathname in both directory trees `practice_q11.sh` does not have to check if it contains the same contents (bytes), just whether both files are the same size (same number of bytes).

For example:
these commands create 2 directory trees named **d1** and **d2** containing 3 files of the same name.

```
$ mkdir -p d1/b/c
$ echo hello andrew > d1/file1
$ echo bye andrew > d1/b/file2
$ echo 1 > d1/b/c/one
$ mkdir -p d2/b/c
$ echo HELLO andrew > d2/file1
$ echo Bye Andrew > d2/b/file2
$ echo 2 > d2/b/c/one
```

The contents of the 3 files are different but their sizes are the same.
practice_q11.sh reports 3 files present in both tree of the same size:

```
$ ./practice_q11.sh d1 d2
3 0 0 0
```

If we change the size of `file1` in the first directory tree **practice_q11.sh** reports 2 files present in both trees of the same size. and 1 file present in both trees but of a different size:

```
$ echo hello everyone >d1/file1
$ ./practice_q11.sh d1 d2
2 1 0 0
```

If we add a file to the second directory tree:

```
$ echo 3 > d2/b/c/three
$ ./practice_q11.sh d1 d2
2 1 0 1
```

If we add a different file to the first directory tree:

```
$ echo 3 > d1/b/three
$ ./practice_q11.sh d1 d2
2 1 1 1
```

Note, the first directory tree contains a file named **b/three** and the second tree contains a file named **b/c/three**.
This is not considered as the file being present in both directory trees, as the relative pathname of each file is different.

> 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.

NOTE:

You do not need to examine the bytes of files, just the files size (number of bytes in the file).
You can however assume files contain only ASCII bytes and are small enough to read their bytes.

Your program does **not** have to consider permissions, modification times, or other file metadata.

You **can** assume the directory trees to be compared contain only directories and regular files.
You **can** assume they do not contain links or other special files.
You **can** assume they do not contain sparse files.

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

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

See `man 1 <program>` for more info on any program

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

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

See `man 1 dash` for more info on any built-in

See `help <built-in>` for *bash* info on any built-in (might not be POSIX-compatible)

You **may not** use non-POSIX-compatible shell features.
You **are not permitted** to use `/bin/bash`, `/bin/sh`, or any other shell.
Make the first line of your shell-script `#!/bin/dash`
You **can** assume that anything that works with the version of `/bin/dash` on CSE systems is POSIX compatible.

You **may not** use `Perl`, `C`, `Python`, `awk` or any language other than shell.

You **may** create temporary files.

No error checking is necessary.

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

```
$ 2041 autotest practice_q11
```

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

```
$ give cs2041 practice_q11 practice_q11.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

Question 12 (8 MARKS)

Write a POSIX-compatible Shell script practice_q12.sh which given a square on the chessboard. prints in successive lines the squares that can be reached by a knight.

A chessboard is an 8x8 square matrix. We label each square as below:

```
a8 b8 c8 d8 e8 f8 g8 h8
a7 b7 c7 d7 e7 f7 g7 h7
a6 b6 c6 d6 e6 f6 g6 h6
a5 b5 c5 d5 e5 f5 g5 h5
a4 b4 c4 d4 e4 f4 g4 h4
a3 b3 c3 d3 e3 f3 g3 h3
a2 b2 c2 d2 e2 f2 g2 h2
a1 b1 c1 d1 e1 f1 g1 h1
```

A knight makes an L-shaped move. It moves either two squares horizontally and one square vertically or two squares vertically and one square horizontally.

For example, a knight at *d4* can move to one of eight squares: *c2*, *e2*, *b3*, *b5*, *c6*, *e6*, *f3* or *f5*.

A move can not take a knight off the chessboard. Hence, a knight on square near the edge of the board will have fewer possible moves.

For example, a knight at *a1* can move only to *c2* and *b3*.

Your program should take one argument: a starting square.

It should print a sequence of lines.

The first line should contain only the starting square

The second line should contain squares a knight can move to from the starting square.

The third line should contain **new** squares a knight can move to from squares listed on the second line.

The fourth line should contain **new** squares a knight can move to from squares listed on the third line.

You program should stop when no **new** squares can be reached.

The squares on each line should be printed in sorted order, separated by a single space.

Match the output format below EXACTLY.

For example:

```
$ ./practice_q12.sh a1
a1
b3 c2
a3 a5 b4 c1 c5 d2 d4 e1 e3
a2 a4 a6 b1 b5 b7 c4 c6 d1 d3 d5 d7 e2 e4 e6 f1 f3 f5 g2 g4
a7 b2 b6 b8 c3 c7 d6 d8 e5 e7 f2 f4 f6 f8 g1 g3 g5 g7 h2 h4 h6
a8 c8 e8 f7 g6 g8 h1 h3 h5 h7
h8
$ ./practice_q12.sh d4
d4
b3 b5 c2 c6 e2 e6 f3 f5
a1 a3 a5 a7 b4 b8 c1 c3 c5 c7 d2 d6 d8 e1 e3 e5 e7 f4 f8 g1 g3 g5 g7 h2 h4 h6
a2 a4 a6 a8 b1 b7 c4 c8 d1 d3 d5 d7 e4 e8 f1 f7 g2 g4 g6 g8 h1 h3 h5 h7
b2 b6 f2 f6 h8
$ ./practice_q12.sh g2
g2
e1 e3 f4 h4
c2 c4 d1 d3 d5 e2 e6 f1 f3 f5 g4 g6 h3 h5
a1 a3 a5 b2 b4 b6 c1 c3 c5 c7 d2 d4 d6 d8 e5 e7 f2 f6 f8 g1 g3 g5 g7 h2 h6 h8
a2 a4 a6 a8 b1 b3 b5 b7 c6 c8 d7 e4 e8 f7 g8 h1 h7
a7 b8
```

NOTE:

Your program must complete in 60 seconds.

Your can assume there is one command-line argument and it is valid square.

No error checking is necessary.

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

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

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

See `man 1 <program>` for more info on any program

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

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

See `man 1 dash` for more info on any built-in

See `help <built-in>` for *bash* info on any built-in (might not be POSIX-compatible)

> You **may not** use non-POSIX-compatible shell features.
> You **are not permitted** to use `/bin/bash`, `/bin/sh`, or any other shell.
> Make the first line of your shell-script `#!/bin/dash`
> You **can** assume that anything that works with the version of `/bin/dash` on CSE systems is POSIX compatible.
>
> You **may not** use `Perl`, `C`, `Python`, `awk` or any language other than shell.
>
> **No** error checking is necessary.

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

```
$ 2041 autotest practice_q12
```

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

```
$ give cs2041 practice_q12 practice_q12.sh
```

To verify your submissions for this activity:

```
$ 2041 classrun -check
```

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.

— END OF EXAMINATION. —

COMP(2041|9044) 23T2: 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