THE UNIVERSITY OF NEW SOUTH WALES
TERM 2, 2022
COMP(2041|9044): SOFTWARE CONSTRUCTION

21T1 Final Exam (22T2 Practice Exam)

— Thursday 18 August 14:00 2022 —
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 at **Thursday 18 August 13:50 2022**, Sydney time.
- You can start working on examination questions at **Thursday 18 August 14:00 2022**, Sydney time.
- You *must* stop working on examination questions at **Thursday 18 August 17:00 2022**, Sydney time. 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.
- You **must not** communicate with any person via any medium during the examination — this includes, but is not limited to, communications via email, telephone, instant-message, talking, hot-air balloon, … — except for COMP(2041|9044) staff, via **cs2041.exam@cse.unsw.edu.au**.
- You **must not** get help from anyone during this exam, except for COMP(2041|9044) staff.
- You **must not** communicate your exam answers to any other person, even after the end of the exam.

  Some students have extended time to complete the exam.
- You **must** ensure that, during and after the examination, no other person can access your work.
- You **must not** place your examination work in a location accessible to any other person, whether they be a student in the course or otherwise. This includes file-sharing services such as Dropbox or GitHub.
- 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 an **open-book examination**: you *may* use your papers or books. You *may not* create or *modify* materials on the Internet, except as would be reasonably required to complete the exam.

**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, worth a total of **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 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 practice_q<N>`

Available Resources: Language Documentation

You may find this **language documentation** useful during this exam:

- 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.

Troubleshooting

If you are having issues working on the exam at CSE, please try the following:

- if you are using VLAB: try logging out and logging back in — you will very likely be connected to a different server, which may make your connection better. If the problem persists, try using SSH instead: instructions at <https://www.cse.unsw.edu.au/~learn/homecomputing/ssh/> or <https://taggi.cse.unsw.edu.au/FAQ/Logging_In_With_SSH/>;
- if you are using SSH: try logging out and logging back in. If the problem persists, try using VLAB instead: instructions at <https://www.cse.unsw.edu.au/~learn/homecomputing/vlab/>
- if you are using VSCode: try disconnecting and reconnecting; or try changing servers from vscode.cse.unsw.edu.au to vscode2.cse.unsw.edu.au.

If you still have problems, contact COMP2041 staff via **cs2041.exam@cse.unsw.edu.au** — we will try to help you fix the problem; and, if we can fix the problem, we may also be able to give you extra time.

Special Consideration

If you experience a technical issue that cannot be fixed during the exam and which means you cannot complete the exam, you may be directed to apply for special consideration. This will require evidence of a problem, and you should take screenshots documenting the problem. For example:

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

You must also contact course staff via **cs2041.exam@cse.unsw.edu.au** as soon as it is clear the issue cannot be fixed.

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.

If you have questions, or you feel unable to complete the exam, contact **cs2041.exam@cse.unsw.edu.au**.

Getting Started

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

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

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 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 (8 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:

Note the input is unordered.
i.e. not sorted in any way.

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

In **practice_q1.sh**, write a shell pipeline that, given student enrollment data in the above format, will output the number of course enrollments by students in the *program* **3711**.

NOTE:

The **second last field** indicates the student's *program* and *stage*.

If a **3711** student is enrolled in multiple courses, all are counted.

Only one line, the number of course enrollments by **3711** students, should be printed.

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

```
$ ./practice_q1.sh < enrollments.txt
4
```

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

```
$ ./practice_q1.sh < more_enrollments.txt
18
```

NOTE:

You **may** assume that the data file always has 6 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_q1.sh`
For example, if your answer to this question is:

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

then `./practice_q1.sh` should contain:

```
$ cat practice_q1.sh
#! /bin/dash
grep "Andrew" | sed 's/^/Hello/' | sort
```

You **may** use the standard UNIX filters.

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

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_q1
```

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

```
$ give cs2041 practice_q1 practice_q1.sh
```

To verify your submissions for this activity:

```
$  2041 classrun -check practice_q1
```

Question 2 (8 MARKS)

Write a Python program **practice_q2.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 questions,
Outputs the number of course enrollments by students in the *program* **3711**.

NOTE:

Note the input is unordered.
i.e. not sorted in any way.

The **second last field** indicates the student's *program* and *stage*.

If a **3711** student is enrolled in multiple courses, all are counted.

Only one line, the number of course enrollments by **3711** students, should be printed.

For example **practice_q2.py** should output this:

```
$ ./practice_q2.py < enrollments.txt
4
$ ./practice_q2.py < more_enrollments.txt
18
```

NOTE:

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` any external `Python` modules.

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** 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.py
```

To verify your submissions for this activity:

```
$  2041 classrun -check practice_q2
```

Question 3 (8 MARKS)

Write a POSIX-compatible Shell script **practice_q3.sh** that lists all the ***happy directories*** in the currect directory.

A ***happy directory*** is a directory that itself contains **2 or more** files or directories.

For example:

```
$ mkdir empty hello goodbye numbers
$ echo "hi there" > hello/file
$ echo "bye" > goodbye/english
$ echo "adios" > goodbye/spanish
$ touch numbers/1 numbers/2 numbers/3 numbers/4
$ ls empty
$ ls hello
file
$ ls goodbye
english  spanish
$ ls numbers
1  2  3  4
$ ./practice_q3.sh
goodbye
numbers
```

NOTE:

A directory is only ***happy*** if it in and of itself contains 2 or more files or directories.

Note a directory that contains a single sub-directory is **not *happy***, even if there are in two or more files or directories in the sub-directory, and the sub-directory is hence *happy*.
In other words, you **do not** have to search recursively down the directory tree to determine if a directory is happy.

For example (continuing from the previous example):

```
$ mkdir d
$ mkdir d/e
$ touch d/e/file1 d/e/file2 d/e/file3
$ ls d
e
$ ls d/e
file1  file2  file3
$ ./practice_q3.sh
goodbye
numbers
$ mkdir d/f
$ ls d
e  f
$ ./practice_q3.sh
d
goodbye
numbers
```

NOTE:

The output order of the ***happy directory*** list doesn't matter.

The current directory **could** contain files as well as directories.

You **can** assume all files and directories in the current-directories and its sub-directories do not start with `.` and do not contain whitespace.

You are **only** permitted to use 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	

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

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

cd	if	while
exit	read	case
for	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_q3
```

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

```
$ give cs2041 practice_q3 practice_q3.sh
```

To verify your submissions for this activity:

```
$  2041 classrun -check practice_q3
```

Question 4 (8 MARKS)

Write a POSIX-compatible Shell script **practice_q4.sh** which takes 2 arguments.

The first argument will contain exactly one positive integer ***n*** as a substring of a larger string.

The second argument will be identical to the first argument *except* the integer ***m*** it contains as a substring may be different.

Your program should print the equivalent strings, one per line, containing the integers ***n*** .. ***m***.

You can assume $n \le m$.

Your program must produce **exactly** the same output as below.

```
$ ./practice_q4.sh aaa723bbb aaa727bbb
aaa723bbb
aaa724bbb
aaa725bbb
aaa726bbb
aaa727bbb
$ ./practice_q4.sh bell11 bell15
bell11
bell12
bell13
bell14
bell15
$ ./practice_q4.sh 6th 9th
6th
7th
8th
9th
$ ./practice_q4.sh 13 14
13
14
$ ./practice_q4.sh ans42er ans42er
ans42er
```

NOTE:

You **can** assume your program is given **exactly 2** arguments.

You **can** assume each argument contains **exactly 1** positive integer as a substring.

You **can** assume the first integer is **not greater** than the second integer.

You **can** assume the non-integer parts of the 2 arguments **are identical**.

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

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

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

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

cd	if	while
exit	read	case
for	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** use the program `seq`.

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_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 practice_q4
```

Question 5 (8 MARKS)

Write a POSIX-compatible Shell script **practice_q5.sh** that can be used to make daily copies of the logging output of a server.

Assume that the script is run in a directory possibly containing files with the names:
`log` , `log.1` , `log.2` , `log.3` , `log.4` , `log.5` , `log.6`
And possibly a sub-directory called `archive`

The file `log` contains server logging output from the past 24 hours.
The file `log.1` contains server access information from yesterday (if it exists).
The file `log.2` contains server access information from two days ago (if it exists).
...
The file `log.6` contains server access information from six days ago (if it exists).

When your program is executed it should do the following:

If the file `log.6` exists, it should be compressed using the *gzip* command.
The compressed file should then be renamed to a name incorporating today's date in the format below, and moved to the `archive/` sub-directory.
For example, if this script is executed on `January 16th 2038` , then the new compressed file should be called `archive/log.2038_01_16.gz`

If the sub-directory `archive` does not exist, it should be created.

All the other log files, if they exist, should have the names updated (shifted):
If `log.5` exists, it should be renamed `log.6`
If `log.4` exists, it should be renamed `log.5`
...
If `log` exists, it should be renamed `log.1`

For example:

```
$ rm -rf archive
$ rm -f log*
$ ls log*
$ echo "selamat" > log
$ ls log*
log
$ cat log
selamat
$ ./practice_q5.sh
$ ls log*
log.1
$ cat log.1
selamat
$ echo "hello" > log
$ ./practice_q5.sh
$ cat log.2
selamat
$ cat log.1
hello
$ echo "namaste" > log
$ ./practice_q5.sh
$ echo "hola" > log
$ ./practice_q5.sh
$ echo "bonjour" > log
$ ./practice_q5.sh
$ echo "guten tag" > log
$ ./practice_q5.sh
$ ls log*
log.1  log.2  log.3  log.4  log.5  log.6
$ echo "ni hao" > log
$ ./practice_q5.sh
$ zcat archive/log.2021_05_06.gz
selamat
$ cat log.1
ni hao
$ cat log.6
hello
```

HINT:

The command *date* has a useful option which allows you to specify its output format.
For this script, `date +'%Y_%m_%d'` should give you today's date in a suitable format

NOTE:

Your shell script **must** produce no output.

Your shell script should not assume the files `log.1 log.2 log.3 log.4 log.5 log.6` exist

Your shell script should not assume the directory `archive` exists

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

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

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

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

cd	if	while
exit	read	case
for	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.

The current directory **could** contain files and/or directories other than `log` , `log.1` ... `log.6` .
Any other files and/or directories should be ignored and untouched by your script.

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 practice_q5
```

Question 6 (8 MARKS)

We have student enrolment data in this familiar format:

```
$ cat practice_q6.0.txt
COMP2511|3713452|Ahmad, Warren                      |3645/2|M
COMP1711|3819596|Hernando, Justin Yeong             |3979/1|M
COMP1511|3953441|Noble, Albert Ka Chuen             |4075/3|F
COMP1521|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
```

Names in this data are in the form: ***Last Name* , *First Names***

We need to translate names to the form: ***First Names Last Name***

Any trailing whitespace after the *Names* should be preserved.

Write a `Python` program **practice_q6.py** that given student enrollment data in the above format from standard input, writes to standard output with the names changed to the form: ***First Names Last Name***

Your program must produce **exactly** the same output as below.

```
$ ./practice_q6.py <practice_q6.0.txt
COMP2511|3713452|Warren Ahmad                      |3645/2|M
COMP1711|3819596|Justin Yeong Hernando             |3979/1|M
COMP1511|3953441|Albert Ka Chuen Noble             |4075/3|F
COMP1521|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
```

NOTE:

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` any external `Python` modules.

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** 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_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 practice_q6
```

Question 7 (8 MARKS)

We need a summary of commits that have been to made to a git repository.

Write a POSIX-compatible Shell script **practice_q7.sh** which prints the date of the last commit, and the number of commits made by each author.
The authors should be listed in decreasing order of numbers of commits.
If authors have made an equal number of commits they should be listed in reverse alphabetical order.

Your program must produce **exactly** the same output as below.

```
$ ./practice_q7.sh
./practice_q7.sh: Not a git repository
$ unzip small_repo.zip
...
$ cd small_repo
$ ../practice_q7.sh
Last Commit Date: Sun May 30 12:00:00 2077 +1000
Number of Commits per Author:
      1 Linus Torvalds
      1 Dylan Brotherston
      1 Alan Turing
$ cd ..
$ unzip large_repo.zip
...
$ cd large_repo
$ ../practice_q7.sh
Last Commit Date: Wed Oct 30 19:34:45 2002 +1100
Number of Commits per Author:
    114 John von Neumann
    110 Dennis Ritchie
    106 Linus Torvalds
    105 Joseph Carl Robnett Licklider
    103 James Gosling
    102 Brian Kernighan
     98 Tim Berners-Lee
     93 Alan Turing
     91 Woz
     78 John McCarthy
$ cd ..
$ unzip empty_repo.zip
...
$ cd empty_repo
$ ../practice_q7.sh
../practice_q7.sh: No Commits
```

NOTE:

Note you are expected to use git commands in your script to extract the infomation.

Do not try to decode the git repo files directly.

You can assume that author names are unique - there are not two people with the same name.

You **can not** assume the current directory is a git repository.

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

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

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

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

cd	if	while
exit	read	case
for	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.

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 practice_q7
```

Question 8 (8 MARKS)

We have a download of Triple J Hottest 100 data in this format

```
poll|position|artist|track
2020|1|Glass Animals|Heat Waves
2020|2|Spacey Jane|Booster Seat
2020|3|Flume|The Difference [Ft. Toro y Moi]
2020|4|Ball Park Music|Cherub
2020|5|Tame Impala|Lost In Yesterday
...
poll|position|artist|track
```

We wish to extract the top 10 songs of each year, along with the artist.

Write a `sed` script **practice_q8.sed** that given input in above format: prints top 10 songs in the format below.

Note, the first and last lines of the data are a header and a footer and should be ignored.

Note, we only want top 10 songs of each year. If the *poll* field is not a year, the song should be ignored.
i.e. *poll*s that are not 4 digits should be ignored.
If the *position* field is not 1..10, the song should be ignored.
i.e. *position*s that are greater than 10 should be ignored.

Your script will be run with **sed -r**.

This provides extended regular expressions, and is what we used in assignment 2.

Your script must produce **exactly** the same output as below.

```
$ sed -r -f practice_q8.sed 2020_only.txt
Heat Waves - Glass Animals
Booster Seat - Spacey Jane
The Difference [Ft. Toro y Moi] - Flume
Cherub - Ball Park Music
Lost In Yesterday - Tame Impala
WAP [Ft. Megan Thee Stallion] - Cardi B
Hyperfine - G Flip
Sending Me Ur Loving - The Jungle Giants
I'm Good? - Hilltop Hoods
Therefore I Am - Billie Eilish
$ sed -r -f practice_q8.sed top_decade_list.txt | tail
Big Jet Plane - Angus & Julia Stone
Rock It - Little Red
Dance the Way I Feel - Ou Est Le Swimming Pool
Plans - Birds of Tokyo
Fall at Your Feet - Boy & Bear
Teenage Crime - Adrian Lux
Fuck You! - Cee-Lo Green
Tokyo (Vampires & Wolves) - The Wombats
Magic Fountain - Art vs Science
Somebody to Love Me {ft. Boy George & Andrew Wyatt} - Mark Ronson & The Business Intl.
$ sed -r -f practice_q8.sed top20_list.txt | tail
Asshole - Denis Leary
Creep - Radiohead
Linger - The Cranberries
No Rain - Blind Melon
Cannonball - The Breeders
Killing in the Name - Rage Against the Machine
Lemon - U2
Go - Pearl Jam
The Honeymoon Is Over - The Cruel Sea
Stone Me into the Groove - Atomic Swing
$ sed -r -f practice_q8.sed master_list.txt | tail
Asshole - Denis Leary
Creep - Radiohead
Linger - The Cranberries
No Rain - Blind Melon
Cannonball - The Breeders
Killing in the Name - Rage Against the Machine
Lemon - U2
Go - Pearl Jam
The Honeymoon Is Over - The Cruel Sea
Stone Me into the Groove - Atomic Swing
```

NOTE:

Your answer **must** be a single sed script.

Your sed script **should** be placed in the file `./practice_q8.sed`
For example, if your answer to this question is:

```
s/.*/Hello Andrew/
42p
```

then `./practice_q8.sed` should contain:

```
$ cat practice_q8.sed
s/.*/Hello Andrew/
42p
```

You can assume **|** characters appear only as field separators.
For example, the track name will never contain an **|**

Your **practice_q8.sed** script will be executed in **exactly** this way:

```
$ sed -r -f practice_q8.sed <data-file>
```

You can assume your script is always executed with a single data file.

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

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.sed
```

To verify your submissions for this activity:

```
$  2041 classrun -check practice_q8
```

Question 9 (9 MARKS)

Write a Python program **practice_q9.py** 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 running **command** as a shell command.

Multiple embedded commands may appear on a single line.

```
$ cat practice_q9.0.txt
contents
of an
example file
$ ./practice_q9.py < practice_q9.0.txt
contents
of an
example file
$ pwd
/home/andrewt
$ echo '<!pwd>' | ./practice_q9.py
/home/andrewt

$ echo '<practice_q9.0.txt>' | ./practice_q9.py
contents
of an
example file
$ cat practice_q9.1.txt
Hello, <!whoami>
You are in the </etc/timezone>timezone.
$ ./practice_q9.py < practice_q9.1.txt
Hello, andrewt

You are in the Australia/Sydney
timezone.
$ date
Tue Jan 16:14:07 2038 AEDT
$ cat practice_q9.2.txt
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.
Here is <!echo practice_q9.1.txt>
<practice_q9.1.txt>
>>> The end. <<<<
$ ./practice_q9.py <practice_q9.2.txt
Lines may have no embedded commands
or several: Tue Jan 16:14:07 2038 AEDT
Tue Jan 16:14:07 2038 AEDT
Tue Jan 16:14:07 2038 AEDT

1 empty file's contents:
3 empty files:

/bin/true prints nothing:

Some commands print no newlines.
Here is practice_q9.1.txt

Hello, <!whoami>
You are in the </etc/timezone>timezone.

>>> The end. <<<<
```

NOTE:

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` any external `Python` modules.

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

You **may** run external programs, using the `subprocess` module.
Eg `subprocess.run("command", shell=True)`
But you **may** only run the commands that have been embedded in your input.

You **may** also run the *cat* 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 **>**.

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_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 practice_q9
```

Question 10 (9 MARKS)

An ***equi-group*** is a set of words, each containing exactly the same letters.

The words in an *equi-group* must contain exactly the same letters occurring the same number of times.

Note: all the words in an *equi-group* must be exactly the same length.

For example:
danger and **garden** are in the same *equi-group* but
reset belongs in a different *equi-group* to **rests**.

Write a Python program `practice_q10.py` which reads lines from standard input until it reaches end-of-input.

Your program will be given lowercase words (only the characters [a-z]), one per line, as input.
The words will be in alphabetical order.

Your program should print all the *equi-groups* for these words.
Each *equi-group* should be printed on a separate line.

The line should start with the number of words in the *equi-group*,
Then the words in the *equi-group* should be printed in alphabetic order, space-separated.

The *equi-groups* should be printed in descending order of size (the number of words in the group).
If two or more *equi-groups* have the same number of words in the group then order them alphabetically.

Note every word in the file will be printed exactly once.

You have been given 3 test data files containing words: `words.tiny.txt` , `words.small.txt` and `words.medium.txt` .

They contain respectively 16, 73 & 10000 words.

Your program must print all *equi-groups* in under 30 seconds for `words.medium.txt` .

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

```
$ ./practice_q10.py < words.tiny.txt
5 caret carte cater crate trace
4 abet bate beat beta
4 ate eat eta tea
1 booster
1 coastal
1 coasted
1 displayed
1 posting
1 roasted
1 roaster
1 rooster
1 singleton
1 undisplayed
$ ./practice_q10.py < words.small.txt
7 pares parse pears rapes reaps spare spear
6 caret cater crate react recta trace
5 bares baser bears braes saber
5 lapse leaps pales peals pleas
5 least slate stale steal tales
5 mates meats steam tames teams
4 abets baste beast beats
4 abler baler blare blear
4 acres cares races scare
4 arced cared cedar raced
3 adder dared dread
3 agers gears rages
3 aimed amide media
3 alert alter later
2 abode adobe
2 abuse beaus
2 aches chase
2 adept taped
1 aback
1 abaft
1 abase
1 abash
$ ./practice_q10.py < words.medium.txt | head
7 pares parse pears rapes reaps spare spear
6 caret cater crate react recta trace
6 caster caters crates reacts recast traces
6 opts post pots spot stop tops
5 alerting altering integral relating triangle
5 arrest rarest raster raters starer
5 bares baser bears braes saber
5 drapes parsed rasped spared spread
5 drawer redraw reward warder warred
5 east eats sate seat teas
$ ./practice_q10.py < words.medium.txt | tail
1 yardstick
1 yearning
1 yell
1 yields
1 yon
1 you
1 youngest
1 yourself
1 zone
1 zones
$
```

NOTE:

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` any external `Python` modules.

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 can assume each line will contain one and only one word.

You can assume each word will contain only lowercase alphabetic characters (a-z).

You can assume a word will contain at most 32 characters.

You can assume your input contains at least one word.

You can assume your input contains no more than 10000 words.

You can assume your input is in alphabetical order.

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 practice_q10
```

Question 11 (9 MARKS)

A ***onesie*** is a pair of words that differ in exactly one letter; for example, "toast" and "roast" or "pink" and "punk" or "shell" and "shelf".

Write a Python program `practice_q11.py` which takes three arguments.

The first argument will be the name of a file containing lowercase words (only the characters [a-z]), one per line.

The second and third arguments will be two words of equal length.

Both words will occur in the file.

Your program should then print the shortest sequence of *onesies* connecting the first word to the second, such that all words are found in the file.

If no sequence of *onesies* exists your program should print the message "No solution.";

If two or more sequences of *onesies* of equal shortest length exist, it may print any one of them.

You have been given a file named `words.large.txt` containing about 30000 words.

Your program must be able to find the shortest sequences of *onesies* in under 60 seconds in `words.large.txt` .

To assist your debugging, you have three shorter files of words, `words.tiny.txt` , `words.small.txt` and `words.medium.txt` .

You may find it useful to create other such small files of words.

Match the output format below exactly.

```
$ ./practice_q11.py words.tiny.txt booster roasted
booster
rooster
roaster
roasted
$ ./practice_q11.py words.tiny.txt coastal posting
No solution
$ ./practice_q11.py words.small.txt bears meats
bears
beats
meats
$ ./practice_q11.py words.medium.txt cat dog
cat
car
tar
tan
ton
don
dog
$ ./practice_q11.py words.large.txt ape man
ape
apt
opt
oat
mat
man
$ ./practice_q11.py words.large.txt live dead
live
rive
ride
rede
redd
read
dead
$ ./practice_q11.py words.large.txt shell files
shell
spell
spill
spiel
spied
shied
shred
sired
fired
filed
files
```

NOTE:

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` any external `Python` modules.

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 can assume your program is given a filename and 2 words as argument.

You can assume each word will contain only lowercase alphabetic characters (a-z).

You can assume both words given as argument occur in the file.

You can assume a word will contain at most 32 characters.

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.py
```

To verify your submissions for this activity:

```
$  2041 classrun -check practice_q11
```

Question 12 (9 MARKS)

Voters can now vote for prime minister by email.

Write a POSIX-compatible Shell script **practice_q12.sh** which given the emails from voters prints which candidate has won the election.

Your program will be given as its first argument, a file containing the names of the candidates one per line.
This file will contain no other lines and there will be no leading or trailing white space.

The remaining arguments to your program will be the names of files each containg a single email from a voter.

For example, the candidates file might have these contents:

```
Adam Bandt
Anthony Albanese
Penny Wong
Scott Morrison
```

And the email from a voter might contain:

```
Date: Tue, 14 Jun 2022 19:54:48 +1000
From: Andrew Taylor <andrewt@unsw.edu.au>
To: votes@elections.gov.au
Subject: vote

Anthony Albanese
Adam Bandt
Penny Wong
Scott Morrison

Thanks for counting my vote,
Andrew
```

Or:

```
Date: Tue, 14 Jun 2022 19:54:48 +1000
From: Andrew Taylor <andrewt@cse.unsw.edu.au>
To: votes@elections.gov.au
Subject: vote

My first choice is

scott MORRISON
adambandt
     anthony    albanese
# I don't like Penny!
PeNNy WonG

:wq
```

For an email to be a valid vote:

- The email must contain the names of all candidates, listed in the order of the voter's choice.
- Every candidate's name must be listed once, and only once, in the email. A vote is not valid if any candidate's name is missing, or if any candidate's name is listed more than once.
- Each candidate's name must appear on line by itself in the email with no other characters except for white space.
- Candidate names must be spelt in the email exactly as spelt in the candidates files, except for differences in case and white space.
  For example, if the candidates file contains "Malcolm Turnbull", a valid vote might contain " malcolmturnBULL ".
- There may be any amount of other text in the email before, after and between the lines listing the names of candidates.

Your program should print a message, following the format in the example below exactly, for each invalid vote.
It should otherwise ignore invalid votes. Invalid votes are not counted in any way

It should otherwise ignore invalid votes. Invalid votes are not counted in any way.

The voting system requires that voters rank all the candidates in order of choice. Initially the first choices are counted, and if one candidate receives more than 50% of the first choices on valid votes, then that candidate is elected.

If no candidate receives more than 50% of the votes, instead the candidate who received fewest votes is eliminated from the election.

If several candidates are tied for the lowest number of votes, they are all eliminated.

After candidate(s) is/are eliminated, any votes for these candidate(s) go instead to the voter's next choice - the next candidate listed in their email who has not already been eliminated .

This process of eliminating the weakest candidate(s) and recounting the votes continues until one candidate receives more than 50% of the vote, or until all remaining candidates are tied.

Your program should print a single line containing the name of the winning candidate. If there is a tie for winner, the names of all candidates who are tied shoudl be printed in alphabetic order.

Your program should produce no other output.

For example:

```
$ ./practice_q12.sh candidates.txt email01.txt
Anthony Albanese
$ ./practice_q12.sh candidates.txt email01.txt  email02.txt email03.txt email04.txt
email02.txt is not a valid vote
Scott Morrison
$ ./practice_q12.sh candidates.txt email01.txt  email02.txt email03.txt email04.txt email05.txt email06.txt
email07.txt
email02.txt is not a valid vote
email07.txt is not a valid vote
Scott Morrison
$ ./practice_q12.sh candidates.txt email05.txt email06.txt email07.txt email08.txt email09.txt
email07.txt is not a valid vote
Adam Bandt
Anthony Albanese
Penny Wong
Scott Morrison
$ ./practice_q12.sh candidates.txt email01.txt email02.txt email03.txt email04.txt email05.txt email06.txt
email07.txt email08.txt email09.txt
email02.txt is not a valid vote
email07.txt is not a valid vote
Anthony Albanese
$ ./practice_q12.sh candidates.txt candidates.txt email01.txt email02.txt email03.txt email04.txt email05.txt
email06.txt email07.txt email08.txt email09.txt email10.txt
email02.txt is not a valid vote
email07.txt is not a valid vote
Adam Bandt
Anthony Albanese
Scott Morrison
```

NOTE:

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

You are **only** permitted to use 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	

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

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

cd	if	while
exit	read	
for	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`, code>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 practice_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 practice_q<N>` :

```
$  2041 classrun -check practice_q1
$  2041 classrun -check practice_q2
...
$  2041 classrun -check practice_q12
```

Remember you have until **Thursday 18 August 17:00 2022**, Sydney time, to complete this exam (not including any extra time provided by ELS conditions).

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

— END OF EXAMINATION. —