Week 03 Weekly Test Questions

Test Conditions

These questions must be completed under self-administered exam-like conditions. You must time the test yourself and ensure you comply with the conditions below.

- You may complete this test in CSE labs or elsewhere using your own machine.
- You may complete this test at any time before **Week 4 Thursday 21:00:00**.
- Weekly tests are designed to act like a past paper - to give you an idea of how well you are progressing in the course, and what you need to work on. Many of the questions in weekly tests are from past final exams.
- Once the first hour has finished, you must submit all questions you've worked on.
- You should then take note of how far you got, which parts you didn't understand.
- You may choose then to keep working and submit test question anytime up to Week 4 Thursday 21:00:00
- However the maximum mark for any question you submit after the first hour will be 50%

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

- manual entries, via the *man* command.
- Texinfo pages, via the *info* command.
- Bash documentation, via the `help` build-in.
- Python documentation, via the `python3 -c 'help()'` command.
- Shell/Regex quick reference
- Python quick reference
- full Python 3.9 documentation

Any violation of the test conditions will results in a mark of zero for the entire weekly test component.

Getting Started

Set up for the test by creating a new directory called `test03` and changing to this directory.

```
$ mkdir test03
$ cd test03
```

There are some provided files for this test which you can fetch with this command:

```
$ 2041 fetch test03
```

If you're not working at CSE, you can download the provided files as a zip file or a tar file.

Test Complete!

Your time for this test has finished. You must submit your work now. You should reflect on how you went in this hour, and discuss with your tutor if you have concerns. You may choose to keep working, but the maximum mark for any questions you submit later will be 50%.

WEEKLY TEST QUESTION:

Printing Student First Names

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 `./student_first_names.sh`

For example, your pipeline should output this:

```
$ ./student_first_names.sh  <enrollments.txt
Heidi
Ian
Ian
Kevin
Liam
Mary
Mary
Mary
Wei
Wei
$ ./student_first_names.sh  <more_enrollments.txt | head
Aaron
Abdullah
Aditya
Adrian
Alvin
Alvin
Bo
Brendan
Brian
Callum
$ ./student_first_names.sh  <more_enrollments.txt | tail
Zeyu
Zeyu
Zeyu
Zeyu
Zeyu
Zhihao
Zhihao
Zihao
Zijian
Ziyang
$ ./student_first_names.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 `./student_first_names.sh` .
For example, if your answer to this question is

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

then `./student_first_names.sh` should contain:

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

You may use any UNIX filters covered in lectures.

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

You must use POSIX shell. No `bash` or `zsh` extensions.

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 `autotest` to run some simple automated tests:

```
$ 2041 autotest student_first_names
```

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

```
$ give cs2041 test03_student_first_names student_first_names.sh
```

WEEKLY TEST QUESTION:

Printing The Most Common First Name

Write a Shell pipeline that, given student enrollment data in the same format as the previous question, outputs the the most common first name for students enrolled in COMP2041 or COMP9044.

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

Your pipeline should print a single line of output.

This line should contain only a single word.

For example:

```
$ ./most_common_first_name.sh  <enrollments.txt
Mary
$ ./most_common_first_name.sh  <more_enrollments.txt
Zeyu
```

NOTE:

If multiple first names are equally most common, you should print the one that is alphabetically first.

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 `./most_common_first_name.sh` .
For example, if your answer to this question is

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

then `./most_common_first_name.sh` should contain:

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

You may use any UNIX filters covered in lectures.

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

You must use POSIX shell. No `bash` or `zsh` extensions.

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 `autotest` to run some simple automated tests:

```
$ 2041 autotest most_common_first_name
```

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

```
$ give cs2041 test03_most_common_first_name most_common_first_name.sh
```

WEEKLY TEST QUESTION:

Printing Students Taking Two Courses

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

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

Put your shell pipeline in a file named `two_courses.sh`

For example, your pipeline should output this:

```
$ ./two_courses.sh  <enrollments.txt
3360582
3860448
3863711
5456732
$ ./two_courses.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
```

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

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

then `two_courses.sh` should contain:

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

You may use any UNIX filters covered in lectures.

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

You must use POSIX shell. No `bash` or `zsh` extensions.

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 `autotest` to run some simple automated tests:

```
$ 2041 autotest two_courses
```

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

```
$ give cs2041 test03_two_courses two_courses.sh
```

Submission

When you are finished each exercise make sure you submit your work by running **give**.

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

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

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Week 4 Thursday 21:00:00** to complete this test.

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

Hint: do your own testing as well as running `autotest`

Test Marks

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

The test exercises for each week are worth in total 1 marks.

Each test is worth 1.7 marks, and will be automarked. Your total mark for the tests component is computed as a sum of your best 6 of 8 test marks.
