CS 251 Outlab 5: Advanced Bash, sed, awk

awk and sed and pretty much all command line tools *that work on Docker* are allowed to be used in this Outlab, unless mentioned specifically in a question. For example in Task 2 in certain subtasks you're only allowed to use specific tools, and no other. All changes from discussion on piazza or other forums will be reflected in bright red, so do keep an eye out. **We may add another question, or adjust the weightage, so please be alert in the initial day or two!** From now (Saturday) on, in case of any ambiguity in the PS of this Outlab, you can generally expect the clarification to be a *simplifying* assumption, i.e. you get the benefit of the doubt. **If there is a corner case the problem statement doesn't make clear how to handle, please handle it in a manner you deem reasonable. If an issue arises, we can discuss in the cribs.**

Extremely sorry for the late clarification. If you're inconvenienced, it is a very minor change in your code. Please note, in Task 2, parts A and B, your script itself should print to the terminal. If you want to test or store the output for further tasks, redirect it into a file through the command line. We will handle this during evaluation. Sincere apologies, once again.

Task 1 - Real Estate Investor (20 marks)

Usage: ./realestate.sh <file1> <file2> <file3>

(Print the above line if the user doesn't enter the required number of arguments, and **exit** with status 1)

Do not hardcode the file names! They will be provided as arguments to your script.

You have bought several properties, and are now looking to rent them out. You have offers from prospective tenants in a CSV (file1, guaranteed to be valid), the record separator is "\n" here: the columns correspond to their name, area of the property (in square feet), the amount of time they intend to rent it (in years), the base rate (per square foot per month), the increment percentage (R) and the maintenance fee, in that order. You'd like to know how much you can get out of each deal.

Here's how it works: every month, you receive 90% of the area times the rate (because TDS). The decimal part of this amount, if any, is **truncated** (this assumption is to make numerical computations in bash easy, also note that the numbers given to you will all be integers). You then pay the maintenance fee, because being the official owner, you are a member of the society.

The base rate is applicable for the starting year. Each year, this is incremented by R%. Again, for simplicity, assume that if the new rate has decimal places, they are truncated. **You need to find out the total income by the end of the lease, for each prospective tenant.**

Having done this, you need to report your computations in file2, which has the format:

Name,Income,Years

Each line corresponds to a tenant. This should be sorted in decreasing order of income, and if there is a tie, in increasing order of years.

Then, in file3, you must print only the names, in the same order as they are in file2.

Overwrite file2 and file3 if they already exist. Do not assume any relation between the paths to these files. 15 marks for getting file2 right, 5 more for getting file3 right.

Task 2 - COAT(Course Organizer and Analyser in Terminal) (50 marks)

Aim: It is always a good habit to plan ahead. This task will help you to create a course visualizer in your terminal and find out how many credits are still left at your disposal
Strictly use **only Sed and awk commands in tasks A,B,E,F**. **You can use loops inside of your sed or awk commands but what you can't do is use it outside of the sed/awk construct .i.e. For task A you can use loops inside of your script.**
Strictly submit only one file per task and all your logic should be inside that file. i.e. you should not use any helper files.
Resources/Inputs:
1) a file containing the list of courses taken by an individual user is given as a CSV file where its first line is having the columns course-code, semester, year, credits, letter grade stored in CSV format .(**allCoursesTaken.csv**)
2) Requirements of no. of credits under an exhaustive list of tags available. It also has color codes mapped to corresponding tags. This information is lifted from here. (you will use it to glorify the boring terminal). (**creditsRequirements.csv**)
3) association of grade with the grade point is given in the file .(**letterGradeToNumber.csv**)
(Assumptions you can or cannot make:-

1. **You can assume that the columns of these input files will remain same in terms of the column names and column orders**
2. **You can assume that letterGradeToNumber.csv and creditsRequirement.csv are exhaustive i.e. there will not be any grade or tag outside of these neither can they be blank inside of allCourseTaken.csv**

3. **You cannot assume that some semester will necessarily have some courses.i.e. You need to display 0.0 in case of no courses being offered in that semester or no courses at all for parts E and F**
4. **You cannot make any helper files and the files that you will be submitting should be exactly the ones mentioned in the submission instructions. You should not submit defineColors.sh and can assume it to be in ./resources/ folder when your scripts are executed.**
5. **You can make as many numbers of finite intermediate files during the execution of your scripts but make sure you remove them before your script terminates. Failing to do so will attract penalties depending on the severity of the divergence.**
6. **You can assume that the colors mentioned in creditsRequirements.csv will match exactly with one of the colors mentioned in defineColors.sh. You should not modify defineColors.sh file.**
7. **Any source you refer from where you copy/read code from should be mentioned in references.txt and in case your code matches with some source not mentioned in references.txt then the matter will be dealt with seriousness**
8. **FF and FR are treated as the same for our purpose. AA and AP too.)**
9. **For consistency, in the folder we have provided, rename “Task2-Resources” as “resources”; place this folder along with your scripts to test. You will finally have to submit only the scripts though.**

Subtasks

A. Create an awk file **viewWithoutColor.awk** which displays the data in allCoursesTaken.csv in a formatted way in the terminal. See sample output to get what does formatting means. Note you need to remove the column “Name” and keep the width of each field to be 20 columns (use printf(“%20s”,$3)). Note the number of highens for the header is 20*(number_of_fields). ~~This should be a generic script and should work on all kinds of tables. Only things hardcoded will be 20 and Name.~~ As long as you match the expected output precisely, you should be ok.
   Command :- **awk -f viewWithoutColor.awk ./resources/allCoursesTaken.csv**
   From here onwards output of task a will be named **outputA**
   10 marks

B. Create a script **viewWithColor.sh** which uses the outputA as input along with the creditsRequirement.csv and uses sed and awk to colorize the output along the lines of the color scheme mentioned in creditsRequirement.csv . Look at the file **defineColors.sh** which is provided to help you in the task. Look at the intended output and do a cat of it to understand what to do. Also, look at this , this link to understand how the coloring actually works.
   Hint:- use a sed command inside of an awk command, or pass commands as awk variables
   Command:- **./viewWithColor.sh outputA ./resources/creditsRequirements.csv**

From here onwards output of task b will be named **outputB**
20 marks
Note:- cat outputB will display the text in coloured format and cat -t outputB can help you figure out how we go on achieving the coloured format.
Note:- Use the line "**source ./resources/defineColors.sh**" in your script.

defineColors.sh is for your convenience, to save you some cumbersome effort. If you find the task difficult, you can hard code the colours, but make sure they match with what is expected.

C. Create a script **viewSemester.sh** that takes in 3 arguments. one as the outputA or outputB(should work on both cases) and the other two as the semester and year. Then it sorts the output w.r.t. the course code and display the same.
Command:- **./viewSemester.sh outputA Autumn 2018**
You can assume there are no "Autumn" or "Spring" as a substring in the name of any course neither does the year match with any course code.
The colors should stay intact for the case of outputB.
5 marks

D. Create a script **viewCourse.sh** that takes as input outputA or outputB (should work on both cases) and another search_string for course_code. You need to display all the courses having the search_string as a substring of the course_code sorted w.r.t. the course code .
Command:- **./viewCourse.sh outputA "CS 1"** (notice the double quotes in the second argument)
The colors should stay intact for the case of outputB. For simplicity, you can assume that the query can only occur as a substring of the course codes.
5 marks

E. Create a script **calculateCPI.sh** that takes as input allCoursesTaken.csv and letterGradeToNumber.csv . For simplicity assume that all courses are considered in CPI calculation (including Minor, Honor and Additional Learning). Output just the CPI correct upto 4 decimal places.(you need to display upto 4 decimal places only)
Command:-
./calculateCPI.sh ./resources/allCoursesTaken.csv ./resources/letterGradeToNumber.csv
5 marks

F. Create a script **calculateSPI.sh** that takes 4 inputs namely allCoursesTaken.csv,letterGradeToNumber.csv,semester and year as input. Output just the SPI correct upto 4 decimal places.(you need to display upto 4 decimal places only).
Command (in one line):-

./calculateSPI.sh ./resources/allCoursesTaken.csv ./resources/letterGradeToNumber.csv Autumn 2016
5 marks

Do compare your output with the expected output, more than just visually! In particular, for this task, you'd want to set your record separators to "\r\n"

A plain `diff` of your output against the expected output should reveal no differences. In particular, please be careful about what order you set the foreground and background colours in!

Submission Instructions

You know the drill. Your submission directory must be called <roll1>-<roll2>, sorted in lexicographical order. Eg. 17D070059-180070035

Compress it into a tarball strictly with the following command (use your roll numbers instead of ours!):

```
tar -zcvf 17D070059-180070035.tar.gz 17D070059-180070035
```

You are encouraged to collaborate on GitHub classroom. Of course, the repo is necessary. The final submission that will be evaluated must be submitted over moodle from the account of the smaller roll number. The deadline is **Tuesday, September 29, 11:59 hours** (it's in the morning this time!). The standard late submission policy applies: if the submission is H hours late (here we take the ceiling function, even 2 minutes counts as 1 hour late), a penalty of 2^H % on your total marks will be applied. So post 6 hours of submission deadline your submission will not be considered.

Here's the expected directory structure, please adhere to it strictly

```
|-- Task1-RealEstate
|   `-- realestate.sh
|-- Task2-COAT
|   |-- calculateCPI.sh
|   |-- calculateSPI.sh
|   |-- viewCourse.sh
|   |-- viewSemester.sh
|   |-- viewWithColor.sh
|   `-- viewWithoutColor.awk
`-- references.txt
```

Important: The first line of your `references.txt` MUST contain an integer: this is approximately the number of hours you took to solve the Outlab. This will not count towards evaluation, but it is a **compulsory** exercise in order to ensure the course runs smoothly (no trolling please). Our autograder may throw an error if you don't do this, and life is short, we would rather do something other than cribs please.