Bangladesh University of Engineering and Technology
CSE 314: Operating Systems Sessional
Shell Scripting Assignment - Submission Organizer and Executor

Change Log

If there is any change in the assignment specification, it will be reflected in the following table.

Description	Updated By	Timestamp
• Assignment declared	Md. Tareq Mahmood	9 June 2023. 3:00 PM.

1. Overview

Suppose you have recently joined as a teacher at an engineering university. You are in charge of evaluating an Algorithm assignment that students have submitted via Moodle. After downloading the submission files, you are in awe □. No matter how many times you have reminded them to submit their files following a standard, some students paid no attention to that. Now you have to organize the submissions and execute them. Luckily, you are an expert in shell scripting. You will use your expertise to automate these tasks and save your precious time.

Your assignment has two parts.

A. Organize the code file into directories by file types and student ID.
B. Execute the codes with test cases and match them with the expected results.

You will be provided with

1. Submission folder (**submissions**)
2. Test case folder (**tests**)
3. Answer folder (**answers**)

2. Task A: Organize

Inside the **submissions** folder, there will be multiple zipped files – one zipped file for each student (see the helping materials section to unzip). Inside the zipped file are the files submitted by the student. The name of the zipped file will be in the following format:

`<Full Name>_<7 digit serial number>_assignsubmission_file_`**`<7 digit student id>`**`.zip`

Students will submit codes in C (.c extension), Python (.py extension), and Java (.java extension). It is guaranteed that one student has submitted only one code file, either in C or Python or Java. There may be other files ending in different extensions. You have to ignore those extra files. The code file of a student can be located inside any directory/subdirectory of the zipped file of the student.

You have to create a new target directory (**targets**). Inside the target directory, there will be three subdirectories - **C**, **Python**, **Java**.

> Suppose, student 1805XXX has written his/her code in C. You have to create a new subdirectory inside *<target directory>/C/1805XXX* and copy the C file inside *<target directory>/C/1805XXX* and rename it to ***main.c***.
>
> Suppose, student 1805XXX has written his/her code in Java. You have to create a new subdirectory inside *<target directory>/Java/1805XXX* and copy the .java file inside *<target directory>/Java/1805XXX* and rename it to ***Main.java***.
>
> Suppose, student 1805XXX has written his/her code in Python. You have to create a new subdirectory inside *<target directory>/Python/1805XXX* and copy the .py file inside *<target directory>/Python/1805XXX* and rename it to ***main.py***.

Refer to the *Match/targets* directory that has been provided to better understand the task.

3. Task B: Execute and Match

Inside the test case folder (**tests**), test cases are stored in the pattern of ***test1.txt***, ***test2.txt***, …, ***testN.txt*** and so on. They are guaranteed to follow such naming conventions.

For each test file, there will also be an accepted answer file inside the answer folder (**answers**), stored in the pattern of ***ans1.txt***, ***ans2.txt***, …, ***ansN.txt*** and so on. They are also guaranteed to follow such naming conventions.

Your task is to compile and run the codes you have organized. Note that Python codes do not require compiling. Refer to the helping materials section to compile and run the code files.

For a C file, the compiled executable file must be named as ***main.out***
For a Java file, the compiled class file must be named as ***Main.class***

You have to store the output files of each test case inside the student's folder inside the organized target folder. The output files will follow the naming convention of ***out1.txt***, ***out2.txt***, …, ***outN.txt*** and so on.

Then, you will match the output files with the corresponding answer files using the command ***diff***. If there are any mismatches between an output file and an answer file, it will be considered as a failure in that test case.

Finally you have to generate a CSV file (inside *<target directory>*, named ***result.csv***) having columns, ***student_id***, ***type***, ***matched***, ***not_matched***. For each student you have to record his/her student ID, his/her programming language (C/Python/Java), number of test cases matched and number of test cases that did not match.

Tentative Algorithm: *(obviously, there are rooms for improvement in this algorithm)*

For each student
    Find the code file type (C/Python/Java)
    Compile if needed
    For each test cases *test<i>.txt*
        Run the code file
        Store the output in *out<i>.txt*
        Match *out<i>.txt* with corresponding *ans<i>.txt* file
    Record how many test cases matched, how many not matched
Generate a CSV file *<target directory>/result.csv*

4. Example

Find the demonstration video here: Shell-Scripting-Assignment-Demonstration.mp4.

Download the "**Shell-Scripting-Assignment-Files.zip**" file from Moodle. After unzipping it, you will find two folders and the following subdirectories.

- ➔ *Workspace*
  - ◆ *submissions* # submission folder
  - ◆ *tests* # test input folder
    - ● *test1.txt*
    - ● *test2.txt*
    - ● *…*
    - ● *testN.txt*
  - ◆ *answers* # accepted output folder
    - ● *ans1.txt*
    - ● *ans2.txt*
    - ● *…*
    - ● *ansN.txt*

- ➔ *Match*
  - ◆ *targets* # this will not be provided during evaluation
    - ● *C* # student who submitted C code
      - ○ *1805121*
        - ◆ *main.c*
        - ◆ *main.out*
        - ◆ *out1.txt*
        - ◆ *…*
        - ◆ *outN.txt*
      - ○ *1805122*

    - *…*
- *Python* # student who submitted Python code
    - *1805128*
        - *main.py*
        - *out1.txt*
        - *…*
        - *outN.txt*
    - *…*
- *Java* # student who submitted Java code
    - *1805130*
        - *Main.java*
        - *Main.class*
        - *out1.txt*
        - *…*
        - *outN.txt*
    - *…*
- *result.csv* # result of each student

The ***Workspace/submissions*** folder consists of zipped files of individual students.

You aim to write a shell script (***organize.sh***) that will be executed inside the ***Workspace*** directory. The script will take 4 mandatory arguments and 2 optional arguments.

Mandatory Arguments (In Order):

1. Path of submission folder
2. Path of target folder (create target folder if it does not exist)
3. Path of test folder
4. Path of answer folder

Optional Argument (In Order):

1. -v
    a. If provided, will print useful information white executing scripts.
    b. Refer to the *demonstration video* to follow exactly what to print and what not to print.
    c. You must only print what has been shown on the video.
2. -noexecute
    a. If provided, will not not perform Task B.
    b. Note that, with this switch, no output files and executable files will be generated.

Sample commands:

```
user@machine:~/Offline/Workspace$ ./organize.sh submissions targets tests answers -v
```

After executing the ***organize.sh*** file, it will generate a new folder inside the ***Workspace*** directory named ***targets***. Your script must generate ***Workspace/targets*** identical to the ***Match/targets*** directory.

```
user@machine:~/Offline/Workspace$ ./organize.sh submissions targets tests answers -v -noexecute
```

```
user@machine:~/Offline/Workspace$ ./organize.sh submissions targets tests answers
```

5. Additional Information

- Each Java file is guaranteed to have **Main** as the main class.
- Each code is guaranteed to compile and run without error/exception.
- You **must** generate a **CSV** file following the exact format that is provided. Order of student ID does not matter. The names of the columns must match.
- You **must** generate executable files, output files following the format. The evaluation will be done by a script and failure to follow the format will result in a penalty.
- If the number of arguments is less than the required number, you **must** print a usage message showing how to use the script.

6. Helping Materials

Unzipping

- Use the command ***unzip*** to unzip files.
- Use ***-d*** switch to unzip to a specific folder.
- Refer to the ***man unzip*** for more.
- Test the command in the terminal before using it inside the script.

Substring Extraction by Pattern

- Run the following commands in a script and see what happens.

```
string=hello.world
echo ${string%.world}
```

Substring Extraction by Index

- Run the following commands in a script and see what happens.

```
string=hello.world
echo ${string:2:1}
echo ${string:2:2}
echo ${string:2:20}
echo ${string: -1}      # mind the space before - sign
echo ${string: -4}
echo ${string:2: -1}
```

Difference between the Two Files

- Use the command ***diff*** to check if the two files are the same.
- Refer to the ***man diff*** for more.
- Test the command in the terminal before using it inside the script.

Arrays

- Run the following commands in a script and see what happens.

```
array=("apple" "mango" "orange")
i=0
echo ${array[$i]}
i=2
echo ${array[$i]}
```

Run a C File

- `gcc file_name.c -o executable_name`
- `./executable_name`

Run a Python Script

- `python3 file_name.py`

Run a Java File

- `javac file_path.java`
- This will create a **.class** file at the same directory as the **.java** file
- `java -cp <directory_of_the_java_file> <name_of_main_class>`

Kill a Running Script from Inside the Script

- `kill -INT $$`

7. Marks Distribution

Tasks	Sub Task	Marks

Task A	Organize Files by Type & ID	20
	Ignore other files	5
Task B	Compile & Run Files	20
	Match Outputs	10
	Generate CSV	10
Target folder matches the specified format exactly		10
Command line argument (Mandatory Arguments)		5
-v Verbose		5
-noexecute No Execute		5
Usage Message		5
Submission matches format (Section 8)		5
Total		**100**

8. Submission

- Create a directory by your 7 digit student name (1905XXX).
- Put your organize.sh file inside.
- Zip the folder, rename it to 1905XXX.zip.
- Submit the zipped folder.

Deadline: 19 June, 2023. 11:55 PM.

9. Plagiarism Policy

- –100% marks will be deducted for plagiarism.
- Work on the problem on your own.