LAB 8 Part III Bourne (again) shell scripts

In this exercise you will be playing with Bourne shell scripts that can solve some problems. Bourne again shell (bash) contains enhancements to Bourne shell and is compatible with Bourne shell. Thus Bourne shell scripts can also be considered Bourne again shell scripts.
Note: our lab provides Bourne Again shell (bash) environment. If you know bash specific syntaxes, feel free to use them. Also, run the scripts under bash (issue `sh` or `bash`).

The purpose of this exercise is to help you gain better understanding about shell script concepts including variable assignment, reading and access, variable input and output, branches and loops etc.

1. Read user input, and do logical comparisons.

In bourn shell, you read user input using utility **`read`**. Navigate to directory `lab8working`. Issue

```
$ read x

We are the world and children

$ echo $x
```

Observe that **`x`**'s value is the whole input line. Now issue

```
$ read x y z

We are the world and children

$ echo $x, $y, $z
```

Observe that **`x`** and **`y`** get the first and 2nd token in the input, and **`z`** gets the rest of input.

In Bourn shell, we do logical comparisons using utility **`test`** or **`[ ]`**. Now issue

```
$ test 3 -lt 4

$ echo $?
```

The above tests if 3 is less than 4. Observe the exit code 0, which means true in Unix. Now issue

```
$ num=3

$ [  $num  -gt  4  ]

$ echo $?
```

No spaces around =

Spaces after **[** and before **]**

The above tests if value of variable **`num`** is greater than 4. Observe the exit code 1, which means false. Now issue

```
$ input="quit"
```

```
$ [ $input = "quitx" ];   echo $?
```

The above tests if variable **`input`** equals to string "quitx", and then displays the exit code. Recall that `;` is a shell meta-character that is used to connect a sequence of commands. Now issue

```
$ [ $input = "quit" ];    echo $?
```

Observe the exit code 0 of the above. Now issue

```
$  [ -f  xFile123.Lab8 ];  echo $?
```

The above tests if **`xFile123.Lab`** is a regular file, and then displays the exit code 0. Now issue

```
$  [ -x  xFile123.Lab8 ];  echo $?
```

The above tests if **`xFile123.Lab`** is an executable file. Now issue

```
$ [ -d  ../lab8working ];  echo $?
```

The above tests if **`../lab8working`** is a directory.

2. Problem B

Copy the classlist file to your current working directory, and do the following exercise.

Download and run the provided shell script **`mygrep.sh`**. Note that the file should have execute permission for the owner. Set the permission if it does not have.

Try to understand the code of the script. And then issue

```
$mygrep.sh
Please enter file to search: classlist
Please enter search key: Wang
```

Observe the output. Then run again, enter search key **`Leung`**, observe the result.

3. Problem C

Download and run the provided shell script **`mygrepArg.sh`**. Make sure that this file has execute permission for the owner. Set it if it doesn't.

Try to understand the code of the script. The program, which takes (at least) two command line arguments,

- first checks if less than two command line arguments are given (how to check?). If yes, then outputs `Error! usage:  ./mygrepArg.sh  filename pattern`
  Note that `./mygrepArg.sh` is not hard coded
- if at least two command line arguments are entered, then checks if the first argument represents an existing file in the current directory. If the file does not exist, outputs `Error! "filename" is not an existing file in the current directory`
  where `filename` is the first command line argument

- if the filename represents an existing file, then displays the two command-line arguments using both `$@` and `$*`. Then conducts the search by invoking `grep $2 $1` (what is `$2` and `$1`)

Run the script with following inputs, and observe the output

```
$ mygrepArg.sh
   Error! usage: ./mygrepArg.sh filename pattern
$ mygrepArg.sh classlistX
   Error! usage: ./mygrepArg.sh filename pattern
$ mygrepArg.sh classlistX Wang
  Error! "classlistX" is not an existing file in current directory
$ mygrepArg.sh classlist Wang
There are 2 command line arguments: EECS2031A Wang or EECS2031A Wang
e*andrew        *********      Wang  Andrew Jr-Yu
haoqwang        *********      Wang  Haoqiu
matt***         *********      Wang  Matthew
weihang         *********      Wang  Wei
$ mygrepArg.sh classlist Leung
There are 2 command line arguments: EECS2031A Leung or EECS2031A Leung
```

4. Problem D

In problem C, the script just executes `grep`, letting the "raw" result of `grep` be displayed, which contains either lines of search result, or just nothing.
Enhance the script for problem C in such a way that

- If the search pattern is not found in file, then instead of displaying nothing, outputs `Pattern "pattern" not found in file "filename"` where `pattern` and `filename` are the 2nd and 1st arguments respectively .

Sample Inputs/Outputs:

```
$ mygrepArg.sh classlist Leung
There are 2 command line arguments: EECS2031A Leung or EECS2031A Leung
Pattern "Leung" was not found in file "classlist"
```

For interested students: sometimes we may want to turn off the raw outputs from the utility call. In order to do this, change the line of utility call to `grep $2 $1 > /dev/null`
This redirects the outputs from `grep` **to a 'black hole' at** `/dev/null`**, where the outputs are absorbed.**

5. Problem E

Variable reading + Branching + Looping in Bourne (again) shell

Download the script `numbers.sh`. Try to understand the code. Then run the script, observe the output.

Sample Inputs/Outputs

```
$ numbers.sh
Enter a number or 'quit': 22
  22 is a positive number
Enter a number or 'quit': 33
  33 is a positive number
Enter a number or 'quit': -1
  -1 is a negative number
Enter a number or 'quit': 0
  0 is zero
Enter a number or 'quit': -13
  -13 is a negative number
Enter a number or 'quit': quit
  Bye bye
$
```

No submissions for Part III.