ECE 364 Lab 01 Handout

Conditionals, Loops, and Basic Commands in Bash

September 1, 2015

Name: _____ ID: ee364_____

1 Instructions

- Work in your Lab01 directory
- Copy all files from ee364/labfiles/Lab01 into your Lab01 directory. You may use the command shown below. It should copy the file `data` and the directory `gradebooks`.
  `cp ~ee364/labfiles/Lab01/* .`
- Remember to add and commit all files to SVN. Also remember to use your header file to start all scripts. **We will grade the version of the file that is in SVN!**
- Name and spell your scripts exactly as instructed. When you are required to generate output, make sure it matches the specifications exactly. Your scripts may be graded by a computer.

2 Gathering data from multiple files

```
SCRIPT NAME:            getStudentData.bash
INPUT:                  From STDIN
OUTPUT:                 To STDOUT
\# OF ARGUMENTS:        1
ARGUMENTS:              course name
RETURN CODE:            0 for success
                        1 for wrong number of arguments
                        2 for invalid input argument
```

2.1 Introduction

Course TAs for ece364 and ece337 have worked long into the night creating files that contain the names of all students enrolled in each class, and their grades for an exam. These files are provided to you in the directory `gradebooks`. You should examine them to get an idea of the exact format. They are formatted as follows:

`<student>,<exam grade>`

Unfortunately, the TAs failed to maintain a central file for each course - students in a particular course are spread across one or more different files. The files are named as `eceXXX_N.txt`, where `XXX` represents the course number, and `N` can be any number. For example, students in ece337 can be found in the two files `ece337_1.txt` and `ece337_2.txt`.

2.2 Implementation Details

To help out your TA, you are instructed to write a script called `getStudentData.bash` that takes in a course name from STDIN, and summarizes student data from all the files for that course.

Your script must meet the following requirements:

- If the number of arguments is not 1, display the message `Usage: ./getStudentData.bash <course name>`, and exit with a return code of 1.
- If the input argument is not one of the two options `ece364` or `ece337`, display the message:
  `Error: course <course name> is not a valid option`
  where `<course name>` refers to the input argument.
- Display the total number of students enrolled in that course in the format: `Total students: <number>`
- Calculate and print out the average exam grade for all students in that course. The message should look like: `Average score: <number>`
- Display the score and name of the student with the highest exam score in that course as follows:
  `<student name> had the highest score of <score>`

2.3 Sample Output

The following session gives an example of the output format that is expected.

```
$ ./getStudentData.bash
Usage: ./getStudentData.bash <course name>

$ ./getStudentData.bash ece364 blahblah
Usage: ./getStudentData.bash <course name>
```

```
$ ./getStudentData.bash cs159
Error: course cs159 is not a valid option.

$ ./getStudentData.bash ece364
Total students: 42
Average score: 79
Dane Ferguson had the highest score of 96

$ ./getStudentData.bash ece337
Total students: 25
Average score: 73
Bernardo Jin had the highest score of 93
```

3 Reading from standard input

```
SCRIPT NAME:            simpleCalc.bash
INPUT:                  From STDIN
OUTPUT:                 To STDOUT
\# OF ARGUMENTS:        3
ARGUMENTS:              operand and two operators
RETURN CODE:            0 for success
                        1 for wrong number of arguments
                        2 for wrong operator
```

3.1 Introduction

You have been asked to write a script called simpleCalc.bash that emulates a simple two-operand calculator that has six main operations:
addition, subtraction, multiplication, division, exponentiation and modulo operation.

The input arguments to your script will be in the order: `<operator> <operand1> <operand2>`
where `<operator>` is the first three letters of one of the above six operations, i.e. it can be either of the following: `add, sub, mul, div, mod,` or `exp`.

3.2 Implementation Details

Your script must meet the following requirements:

- If the number of arguments is not 1, display the message
  `Usage: ./simpleCalc.bash <operator> <operand1> <operand2>` and exit with return code 1.
- If the first input argument, the operator, is not one of the six functions described above, display the message `Error: invalid operator.` and exit with return code 2.
- You do not have to validate the next two input arguments. You can assume they will always be integers.
- Once input validation is completed, perform the arithmetic operation. For example, if the input is `add 2 4`, your calculator must compute the value "2+4".
  HINT: The exponentiation operator in Bash is a double asterisk (**).
- The final output of your script should look like:
  `<operand1> <operator> <operand2> = <result>`.
  where `<operator>` is any of the following: `+, -, *, /, %, ^`. So for the example above, your script should display the following to `STDOUT`: `2 + 4 = 6`

3.3 Sample Output

The following session gives an example of the output format that is expected.

```
$ ./simpleCalc.bash add 2
Usage: ./simpleCalc.bash <operator> <operand1> <operand2>
$ ./simpleCalc.bash 5 6 7
Error: invalid operator.
$ ./simpleCalc.bash add 6 7
6 + 7 = 13
$ ./simpleCalc.bash exp 4 2
4 ^ 2 = 16
$ ./simpleCalc.bash mod 3 5
3 % 5 = 3
```

4 Using a script within another script

```
SCRIPT NAME:            advCalc.bash
INPUT:                  From file
OUTPUT:                 To STDOUT
\# OF ARGUMENTS:        1
ARGUMENTS:              file name
RETURN CODE:            0 for success
                        1 for wrong number of arguments
```

4.1 Introduction

You have been provided with the file `data` that contains a list of arithmetic operations to perform. Unfortunately, some of these lines have errors, and so you must detect them. View the file to get an idea of the content. Since the file contains too many arithmetic operations to handle individually, you decide to write a script called `advCalc.bash` that reads from the file, performs input validation, and feeds the input into `simpleCalc.bash`.

4.2 Implementation Details

Below are the requirements for `advCalc.bash`:

- Your script **MUST** execute `simpleCalc.bash` from Part 3. The return code from the execution of `simpleCalc.bash` should be checked.
- For each line read from the input file, your script must display:
  `<line read from file>: <result obtained from simpleCalc>`
  See the example provided below.
- If the return code from the execution of `simpleCalc.bash` is nonzero, your script must print out the word `Error` in place of the result.
- For example, if the input file had the following three lines:

  ```
  add 2 2
  exp 6 4 2
  mul 2 2
  ```

  Then your script should print out:

  ```
  add 2 2: 2 + 2 = 4
  exp 6 4 2: Error
  mul 5 3: 5 * 3 = 15
  ```

4.3 Sample Output

The file `gold_output` has been provided to you. You may check the final output of your script against this file.