COMP-206
Software Systems

Assignment #2
Due: February 19, 2017 on myCourses at 23:30

Question 1: Engineering skills through Bash programming [8 points]

Every programmer when developing a new project has techniques they would like to standardize. These techniques represent best practices they have learned over the years. This question asks you to write a Bash script that incorporates some of these common practices. The engineering practices in question are the project's directory structure and a script that helps when compiling.

Create a Bash script called NewProject that does the following when executed:

- It's command-line syntax is:

  ```
  ./NewProject path project_name
  ```

  - The user must type in the project's name and path as command-line arguments to the script. These two parameters are separated by a space. In other words $0=NewProject $1=path $2=project_name. The user must enter NewProject and the user provides their own values for the arguments path and project_name.
  - If the project name argument is missing then an error message is displayed saying:

    "Project name is missing. Type in: NewProject path project_name".

    The script then terminates.
  - The path is optional. If the user provided a path then the path must be a standard Unix path name. If the path is malformed then the following error message is displayed:

    "Your path name is malformed. Type in: NewProject path project_name".

    The script then terminates. A malformed path means that it cannot be used to express a path.
  - The project name must be one word.
- The script uses the project name to create a sub-directory based on the path provided. If no path was provided then the directory is created within the current directory. The created sub-directory name is the project name.
- Within that sub-directory the script constructs a best practice directory structure. Specifically it creates the following sub-directories within the project name directory: docs, source, backup, and archive.
- Finally, within the newly created source directory, NewProject will generate (write) it's own compiling script called compile! The user will then be able to use that compiling script while working on their C projects! This is how you should build that compiling script. Note, you are not copying a file from another location. You are generating the script from programming statements contained within the NewProject script.

- Hint: `echo "some text" >> filename` will append "some text" to the given file name. If the "some text" is a Bash instruction then the above statement appends that instruction to the text file. An entire script can be created this way from within a Bash program!
- You will need to `chmod` the compile script you generated to make it executable. Do this also from NewProject.
- The compiling script's command-line syntax is:
  `compile -o executable_name file_name(s)`
  - The name of the generated script is <u>compile</u>.
  - It has on **<u>optional</u>** switch -o that is paired with an argument called <u>executable_name</u> in the example above. The user will provide their own name for executable_name at the command-line (through the positional variables).
    - Your compile script uses the gcc command. The -o executable will be used as the `gcc -o output_file_name` (as seen in class).
    - If the user did not provide a -o switch for the script then your gcc command must produce an a.out file.
  - The last argument is called <u>file_name(s)</u> and represents a list of space separated C source file names. There can be one or more (any number actually). If there are none, then an error message is displayed:
    "You are missing file names. Type in: compile -o executable_name file_name(s)".
  - The compile script will do the following when compiling: (1) before it compiles it will copy all the source file(s) into the backup sub-director. (2) Then, it will use gcc and optionally the -o switch to compile the program. But it will redirect the errors into a text file called <u>errors</u>, overwriting (without prompt) any previous text file with that same name. (3) The script ends by using "more" to view the error file.

Test your scripts from question 1 by using them to write the two C programs in question 2 and 3 below.

<u>Question 2: Guess [4 points]</u>

There is a game where you have to guess what two thirds of the average of all guesses will be. It works like this: 10 people guess a random number between 1 to 50. This data is stored in a file as the initial data available to begin playing. Now a user runs the script and is prompted for input where he enters a number. If the number is within +/-10% of the 2/3rds of the average of input data in the file, the player wins. The user can guess up to 3 times before the script terminates and prints the number of guesses the user took to guess the right number and how the user compares to other players.

[1 point]

A) As you do not have initial data, write a function to generate this data through the use of a random number generator. This function should only run the first time when the game has not been played by any player so far.

[1 point]

B) After each guess, the data should be updated to include this guess in the data file and replace one of the random numbers you generated. (Each guess will replace one random generated number).

[1 point]

C) Keep track of the average number of tries a player took for the correct guess for 5 last players. Again you could start with a random number but after each play, this data should be updated to reflect the last play. [Ex: 5 players took tries: 2,1,3,2,2. So average is 2]. You should print this message when user is done with his tries or has correctly guessed the right number.

[1 point]

D) Since the user shouldn't be able to read the input data of other users, you must encrypt it in any manner you desire but it should be readable by your script. (Hint: Use the algorithm in Q1 to shift/add to numbers)

Expected Format:

Game script : playgame.sh

Game script file: Data.pg (Should be at the same directory as playgame.sh)

Sample Run:

1.) User runs script. [./playgame.sh]

2.) User is prompted for guess [Guess Number?]

3.) User enters number [10]

4.) If number is not within 10%, he is prompted again [Incorrect guess. Try again.]

5.) User enters number [20]

6.) User is shown message for a correct guess [Well Done. You took 2 tries to guess. Average tries is 2]

Question 3: Cryptography in C [8 points]

In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.

Example: The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. For instance, here is a Caesar cipher using a left rotation of three places, equivalent to a right shift of 23 (the shift parameter is used as the key):

```
Plain:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:   XYZABCDEFGHIJKLMNOPQRSTUVW
```

When encrypting, a person looks up each letter of the message in the "plain" line and writes down the corresponding letter in the "cipher" line.

```
Plaintext:  THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD
```

Deciphering is done in reverse, with a right shift of 3.

A) Implement a C Language program to implement this cipher.

The program takes two inputs:

1) File name that user wants to encrypt.
2) The key the user plans to use (For the assignment, we will restrict it to a left shift between 1 and 25. Ex, a 3 means shift left by 3 places).

The program replaces the unencrypted contents of the input file with the newly encrypted text. The contents will only be text using alphabets with spaces as in the above example. We will not test it for numbers or special characters.

B) Implement a second C Language program that deciphers this encrypted file:

The program takes two inputs:

1) File name that user wants to decrypt.
2) The key to decrypt it (the correct input is the key supplied while encrypting it above)

   Expected format:

Encryption

[File to be encrypted: Contents]

classifiedfile.txt : THE QUICK BROWN

[Execution Format: ./encipher.sh <filename with path> <key>]

```
./encipher.sh classifiedfile.txt 3
```

[File to be encrypted: Contents]

classifiedfile.txt : QEB NRFZH YOLTK

Expected format: **Decryption**

[File to be decrypted: Contents]

classifiedfile.txt : QEB NRFZH YOLTK

[Format: ./decipher.sh <filename with path> <key>]

```
./decipher.sh  classifiedfile.txt 3
```

[File to be decrypted: Contents]

classifiedfile.txt : THE QUICK BROWN

Note: It's important that you name your programs the same as in the given format with the same number of arguments and type.

We will test your programs using a different filename and key.

FOR THE GLORY

Glory questions are given out without help. You must solve this on your own. It is also optional since no points are awarded. The TA will however look at your solution and give you comments if you submit it with your assignment. The TA will also select the best solution from the group of students they were given to grade and post it in the announcements section of myCourses. They will post only one solution. If there are more than one "best solutions" then they will post the first one they encountered. Each TA will do this so multiple solutions will be posted... assuming anyone qualifies...

Question 3 asks you to implement Caesar's cipher which is quiet easy to crack. Another variation and a slightly harder but dramatically more effective variation is the double-Caesar cipher. It works just like the Caesar cipher, but each letter in your message gets shifted by a different amount based on another string of text called the key.

Each letter of the key indicates number of shifts to advance: a is 0, a b is 1, and so on. E.g: if your original message is "assignment" and your key is "add", the "a" in "add" means that you shift the first letter of " assignment " 0 letters, the "d" in "add" means you shift the second letter of " assignment" by 3 letters, and repeat the pattern.
Example:

Input Text:	assignment
Key(repeated)	addaddadda
Output Text	avvijomhqt

Implement this variation keeping the original format in Q1 with only replacing the number(key) in the execution format with a alphabetic key.
E.g. :
```
./encipher classifiedfile.txt <key>
./encipher classifiedfile.txt add
```

HOW IT WILL BE GRADED

Points removed for bad practices:

- -1 for not following instructions
- -1 for not indenting, spacing, and/or commenting
- -1 for not using good variable names

This assignment is worth 20 points:

- Question 1 is worth 8 points
    - +4 NewProject
    - +4 Compile
- Question 2 is worth 4 points
- Question 3 is worth 8 points
    - +2 Encrypting
    - +2 Decrypting
    - +2 Command-line arguments
    - +2 File processing
- Glory question comments from TA
    - Best of glory solution will be posted on the announcements section of myCourses by each TA for the group of students they are grading (if anyone qualifies).