COMP-206
Software Systems
Assignment #2
Due: February 17, 2014 on My Courses at 23:55

The purpose of this assignment is to further increase your Bash programming skills, show you some command-line engineering skills, and introduce you to C programming.

Question 1: Bash Programming

Every programmer and engineer 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 practices. The engineering practices in question are the project's directory structure and a script that helps in compiling.

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

- It's command-line syntax is: `NewProject project_name`
  - The user must type in the project's name as the command-line argument to the script.
  - If the project name argument is missing then an error message is displayed saying "Project name is missing. Type in: NewProject project_name".
  - The project name does not have a path.
  - The project name must be one word. Your script must check for this too. It must display this error message: "Your project name must be a single word. Type in: NewProject project_name".
- The script uses the project name to create a sub-directory within the current directory. The sub-directory's name is the same as the project name.
- Within that sub-directory the script constructs a best practice directory structure. Specifically in creates the following sub-directories below the project name directory: docs, source, backup, archive.
- Finally, within the newly created project's 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!
  - Hint: `echo "some text" >> filename` will append "some text" to the given file name. If the text is a Bash instruction then you've added that 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 file(s)`
    - The name of the generated script is compile.
    - It has on optional switch -o that is paired with an argument called executable in the example above. The user will provide their own name for executable at the command-line.

- Your compile script uses the C gcc command. The -o executable will be used as the `gcc -o output_file_name` (as seen in class).

- The last argument is called file(s) 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 file(s)".
- The compile script will do the following when compiling: (1) before it compiles it will copy all the 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 errors, 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.

Question 2: C Programming

At the command line, use question 1 to create a sub-directory called ass2. In the source sub-directory of ass2 vi a file called ascii.c and in that file write the following program:

Write a C program that asks the user for single characters, one at a time, and prints out their ASCII value until they input the number zero.

Note: It is best to not use the char type, instead put your characters in an int.

This program will have two ways to input characters: (a) from within the program, and (b) from the command-line prompt . See the examples below:

(a) $./ascii

And your program should output:

```
Welcome to ASCII:
----> f
ascii: 102
----> 3
ascii: 51
----> 0
ascii: 48
Done.
```

Notice in the output above the program begins with a title and then shows a prompt. The user inputs a single character (in the example user input is in bold). The program responds with the message 'ascii:' and displays the ASCII value, and repeats the prompt. The user then continues to input more characters until they want to quit. At that point the user inputs the character zero, its ASCII value is displayed, and the program terminates with the message "Done.".

(b) $./ascii f 3

And your program outputs:

----> **f**
ascii: 102
----> **3**
ascii: 51
Done.

Notice that the output is similar to what was displayed in (a) except that the zero was not needed. The user can enter any number of arguments at the command-line. The program processes all the arguments one at a time displaying their result, as above. The program then terminates.

Your program should look and behave as described above.

Question 3: Caesar Cipher

Create a C program that prompts the user for a sentence of no more than 100 characters, check for this. It then prompts the user for an integer number. The user can input a positive or negative number. It can be any number. The program will use Caesar cipher to cipher the (up to) 100 characters entered by the user. Caesar cipher is circular therefore ciphering a text with a positive number and then ciphering the ciphered text with the negative value of the same number will return the ciphered text back to its original letters. The program must be case sensitive. It must also ignore all none alphabetic characters. In other words, it will leave those characters un-ciphered.

Note: Only letters are ciphered. Letters are ciphered back to letters. Caesar used integer numbers to shift letters. For example the letter 'A' shifted by 2 would be 'C'. The letter 'C' shifted by -1 would be 'B'. The letter 'Z' shifted by 4 would be 'D'. The latter 'A' shifted by -2 would be 'Y'. Notice how they wrap around. The letters are also case sensitive. Upper case letter cipher to upper case letters and lower case letters cipher to lower case letters.

Your program should behave like this:

$./caesar
Sentence: Bob
Number: 1
Cipher: Cpc
Done.

$./caesar
Sentence: I am 12!
Number: 1
Cipher: J bn 12!
Done.

$./caesar
Sentence: J bn 12!

Number: -1
Cipher: I am 12!
Done.

HOW IT WILL BE GRADED

Points removed for bad practices:

- -1 for not following instructions
- -1 for not indenting
- -1 for not using good variable names

This assignment is worth 20 points:

- Question 1 is worth 10 points
  - +5 NewProject
  - +5 Compile
- Question 2 is worth 5 points
  - +2 from within the program
  - +3 from the command-line
- Question 3 is worth 5 points
  - Case sensitive
  - Wrap around
  - Only letters