Lab 4. Code Blocks, Functions, Scope

Goal:
To practice writing C code using functions, multiple source files and experimenting with Scope. We'll also use the Unix command "ar" for archiving object files (.o) into a library (.a)

Make sure that you understand each step of the process, use the Lecture notes, man pages, …

Part 1. Write code per the specification below:

1. Write a program that will perform math functions and display the results based on the user input of an integer value
2. Prompt the user for a single integer value and use the **scanf**() function to get the user input**.**
3. Use a loop to continually prompt the user for another number to process, let the number 0 indicate **Exit** or **Quit** the program.
4. Implement the following two math functions in two separate **.c** files:
    1. **fun1.c**: Contains 1 function:
        - **fun1()**: Given an integer value, return the square of the given integer
    2. **fun2_3.c**: Contains 2 functions:
        - **fun2()**: Given an integer parameter, return the next integer (i.e. add 1)
        - **fun3()**: Given an integer parameter, return half of that number as a **float**

- Create a **main.c** which has the **main**() function which calls **fun1**(), **fun2**() and **fun3**().
- Create a **funs.h** to contain the **extern** declarations of the functions used in **main**().
- The **main**() function should:
    - Accept Command Line Parameters (i.e **main**() will need to identify parameters)
    - Print the following line one time before the loop:

      `Running <prog name> with parameter "<first param>"`

      Where <*prog name*> is the name of the program and <*first param*> is the first parameter provided on the command line in double quotes:

      e.g. running the program as "./a.out foo" would result in:

      `Running program a.out with parameter "foo"`

    - Prompt the user to input an integer value in a loop
    - Call each of the two math functions to calculate the results for each loop iteration. Print the results as:

      ```
      The square of <num> is <result of fun1()>
      The next num is <result of fun2()>
      The half of <num> is <result of fun3()>
      ```

    - Exit the loop when the user enters a zero (0), do not print the result lines for 0

    Use **%i** for **int** format for the input (understand the difference between **%i** and **%d** for getting integer input with **scanf**()). Use **%6.2f** for **float** value output

A hint: To get a float result when dividing in fun3(), you need both numbers to be floats.

E.g. "<float num> / **2**" will drop the precision, instead use: "<float num> / **2.0**"

But try it with both to see how it behaves with a 2 vs a 2.0

Part 2. Build and Run the Executable:

Build the executable with the following steps:

1. Compile each of the 3 **.c** files using the **cc** command flag "**-c**" to create object files for each
2. Archive the **fun1.o** and **fun2_3.o** files (only) into an archive named **mathFuns.a** (*see lecture*)
3. Build the executable using the cc command line**: cc -o doMath main.o mathFuns.a**

   ***Note**: each time you change something in a .c, you need to do all steps (cc -c, ar, cc) to include that change in the doMath executable*
4. Run **./doMath** yoyo
5. When prompted, enter the values **5, 27, 8, 0** at subsequent prompts (i.e. hit return between the input numbers)
6. Observe the results

Part 3. Regular Expression Practice:

Practice using regular expressions with the **egrep** (or "grep -E") command.
Using the provided file Gettysburg.txt:

1. Find the line(s) containing the fixed string "liberty"
2. Find the line(s) that start with a 'd' or 't' and end with a 'd'
3. Find the line(s) that contain the word "this" then the word "under"
4. Find the count of line(s) that end with a period or a comma
5. Find the line(s) that contain the word "can", not lines that have "cannot"
   (look at using an egrep parameter for #5)

PREPARE YOUR FILE FOR GRADING.

When all is well and correct, and you are still on our Linux machine…

- type: **script StudentName_lab4.txt** To capture the output of this session
- Repeat Part 2 and Part 3 To compile and run the program
- Type: **rm -f *.o *.a doMath** To remove your build files
- Type: **zip -rv lab4Files.zip *** To create a zip file of all project files
- Type: **unzip -tv lab4Files.zip** To verify the zip archive and see it has all the files
- type: **exit** To end this script session.

If necessary, copy your 2 files to be accessible to your browser for upload into Canvas.

Turn in your work.

Go to Canvas, answer the questions and turn in your 2 files:

1. lab4Files.zip …the source code archive for this lab
2. StudentName_lab4.txt …the script output file