Prelab: Before starting your m-file, read through this entire document, then create a pseudocode flowchart of what your program will do using the standard symbols given in Section 8.2 of your textbook.

Lab: Create a new m-file called **taylor_series.m** that will contain a user-defined function. The function will have the function name, inputs, outputs, and **help** command information as shown.

```
function [converges, iter] = ...
    taylor_series(func, x, error_ub, max_iters)
% Performs the Taylor series expansion for either the sine of x or
% for Euler's number raised to x.  Function returns whether or
% not the function converges within the user-defined error_ub
% within max_iters iterations.
%
% Input:
%     - func: A string describing which function to do the Taylor
%           series expansion on.  ('sin' for sine(x), 'exp' for e^x)
%     - x: Input argument that the function will operate on.  (must be
%           >= -50 and <= 50)
%     - error_ub: upper bound for error, or maximum allowed error
%           (absolute value) in order to consider the
%           series expansion converged. (must be >= 1e-12)
%      - max_iters: User-defined maximum iterations to try before
%           giving up. (must be positive integer <= 150)
%
% Output:
%     - converges: equals 1 if series converged to an error <=
%           error_ub within max_iters or less, else equals 0.
%     - iter: if converges is 1, equal to number of iterations it took
%           to converge within error_ub, else equals 0.
%
```

Underneath this, include a block comment with your name, the date, and the lab assignment number. Underneath that, create a function that fulfills the given description.

Your function should verify that all four of its input variables have values as specified in the description, otherwise, use the **error** command to tell the user which variable is invalid and give the constraints. For example, if a user tries to call the function with **x** set equal to -60, this should trigger the following command.

```
error('x must be >= -50 and <= 50')
```

Your function should use a **switch** statement to verify the contents of the **func** variable.

The Taylor series for our two functions are below.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ...$$

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - ...$$

Note that as **n** increases, each successive term gets smaller in absolute value. Knowing this, we can assert that if, for instance, the 10^{th} term in a Taylor series is 0.005, then 10 iterations of the function will bring the sum of terms 1 through 10 to be within +/- 0.005 of the actual answer. In other words, 10 iterations yields an answer that has an error with absolute value of 0.005 or less.

Your function shall use a **while** loop to compute each of these terms until the latest term's absolute value is equal to or less than **error_ub**. If/when the latest term in the Taylor series is within this boundary, use an **if/break** statement to exit the **while** loop. Otherwise, the **while** loop should keep running until it has been run **max_iters** times.

Note that your **taylor_series** function will not be computing the sum of of these terms, nor will it be approximating the actual value of $sin(x)$ or e^x.

You might find to be somewhat helpful. Some other recommendations:

- Example 9.6 in your textbook shows how to use an **if/break** statement along with a flowchart of its algorithm.
- Start working with your m-file as a regular script (not a function) with hard coded input values. This way you can easily run and re-run it for testing.
- Get either the e^x or the $sin(x)$ capability working before adding the other capability.
- Once both of these capabilities are working, do the input validation for all of the input variables.
- Test your code often!
- Wait until you have verified your function works, along with its input validation, before changing it into a function.
- Now call your function from a separate m-file. It is easier to change values and re-run this way,v versus calling it from the command line each time.
- Verify your results. Is your algorithm actually converging on a real solution? You can add in a variable that sums each term's results and then compare it to the Matlab functions' results for the two respective operations.

Your instructor may provide some sample output file called **ENGR114-Lab06-sample_output.txt**. Once you are satisfied that your file meets all the requirements, suppress all screen output, save the file, and submit it to appropriate D2L folder.

