ECEN 5823 Spring 2025
Internet of Things Embedded Firmware
Assignment #0.5 - Circular Buffer

Objective: In many applications, memory space is a fixed and finite resource, and as such requires us to manage memory usage wisely in our implementations. Circular buffers (sometimes called "*circular queues*" or "*FIFOs*" (first-in first-out, pronounced ˈfīˌ fō)) are a very common technique. Circular buffers are often used in systems where a "*producer*" generates data at a different rate than a "*consumer*". These are often called "*elasticity FIFOs*". In this assignment you will learn to construct a circular buffer in C. ECEN 5813 Principles of Embedded Software (PES) is a prerequisite for this course, and covers circular buffers. If you haven't taken PES or your are taking PES concurrently, this assignment exists to ensure you gain crucial experience in building circular buffers. Circular buffers will be required for a later Bluetooth assignment, so it is imperative that you get experience now. If you already know how to build circular buffers, then this assignment should be a breeze.

Note: This assignment is a 1-off and does not depend on previous Simplicity Studio assignment code. For this assignment you can use:

- Simplicity Studio and run the code on your Gecko.
- Another development environment like Microsoft Visual Studio, or Xcode on a Mac.
- Some command line environment like linux, using gcc.

Instructions:

1. Start by creating your GitHub Classroom assignment repository using the assignment link at https://classroom.github.com/a/iNjEDyL6 .

2. Clone this repository into your local machine.

Linux example:

```
% cd ~/my_code
% git clone https://github.com/CU-ECEN-5823/ecen5823-assignment05-cchoi22915.git
```

cchoi22915 is my GitHub username. A **new directory** will be created in ~/my_code/ called ecen5823-assignment05-cchoi22915/

Populate the local repo with code from the assignment0.5 starter code repository:

```
% cd ecen5823-assignment05-cchoi22915
% git remote add assignments-base https://github.com/CU-ECEN-5823/ecen5823-f21-circular-buffer.git
% git fetch assignments-base
```

```
% git merge assignments-base/master [1]
% git push origin master
```

3. You will see that there are 2 starter source files: **circular_buffer.h** and **circular_buffer.c**. Edit these files to implement your circular buffer. An **example_main.c** has been provided to you such that you can see how a test bench can be created. Use example_main.c if you want to, or develop your own main.c functionality.

4. Create a main.c file (and main() function) to call and test your circular buffer (we sometimes call this a test bench). There are 6 functions to implement:
    a. nextPtr() - this takes a current pointer value and computes the next pointer value.
    b. reset_queue() - this should reset your queue to "time 0", no entries.
    c. write_queue() - this writes an entry into the queue. When writing, we pass in 3 parameters: charHandle, bufLength and buffer, to be written to the entry pointed at by the write pointer. Decide on how you want to handle the "full" condition.
    d. read_queue() - this returns an entry from the queue.
    e. get_queue_status() - this returns the pointers, and the full and empty conditions.
    f. get_queue_depth() - this returns the number of written entries currently in the queue; i.e. How many entries are currently in there?

    Things to note in your design:
    QUEUE_DEPTH is used to set the number of entries in your queue.
    My test bench (the autograder), that will test the functionality of your code, can handle 2 designs:
    - 1- Designs that use all of the entries, #define USE_ALL_ENTRIES (1) - which requires write pointer suppression, or
    - 2- Designs that leave 1 entry empty, #define USE_ALL_ENTRIES (0)

    Any other design type is not handled by my test bench, you'll end up with a poor grade. Select one of the 2 design options for your implementation.

5. Test your code throughly. Cases to cover:
    a. Fill your queue and then try to write to a full queue. When the queue is full, the write operation must do nothing - yes that's correct, the write data is lost in this case, i.e thrown away. The write pointer should not advance when full.
        i. Test full when the write pointer > read pointer, and
        ii. Test full when the read pointer is > write pointer
    b. Read from an empty queue. No data should be returned, the read pointer should not advance when empty.
    c. Write and then read back a number of values.
        i. Test full when the write pointer > read pointer, and
        ii. Test full when the read pointer is > write pointer

[1] Some Windows users get a "Filename too long" error on the merge command. If this happens try these to commands prior to the merge:

```
% git config --system core.longpaths true
% git fsck --name-object
```

Deliverables:

- Submission to **github** via your repository setup with the classroom link at above. Use the following git command to submit your assignment:
    - git push origin master
    - The only files that are required to be submitted to GitHub are:
        - **circular_buffer.h** and
        - **circular_buffer.c**.
    - **There is no requirement to submit the example_main.c or any other file**. You can, if you'd like, submit other files, they will not impact our testing or your grade on this assignment. The autograder will only use your **circular_buffer.h** and **circular_buffer.c**.
- Verify your ecen5823-assignment05-<username> repository contains your latest/ intended code. It is your responsibility to push the correct version of your code. Use a web browser to verify the correct version of your code has been uploaded to GitHub.
- In **Canvas**, submit the URL to your github repository and in **GitHub** create a tag of your GitHub submission prior to the due date with the text of A05-final. The date and time of the tag creation is what we use to determine whether your submission was on time or late. These 2 steps will complete your assignment submission.
- Your assignment will be auto graded by a test bench I have constructed that exercises your implementation and renders a numeric score (grade).