Lab 1 Pre-lab

This lab's primary focus is to ensure students feel comfortable in the development environment and can write, build, and flash code to the microcontroller. It will also act as a refresher to some C concepts.

How to read the lab write-ups:

Each lab will have an appropriate pre-lab write-up informing students about specific things within the lab and lab write-up. The prelab must be read and understood before attempting the lab. Failure to do this will result in you being confused and potentially taking bandwidth away from the instructional staff to answer a question that the prelab would have addressed. Don't be that person. You do NOT have to do the instructions given below, they are just to act as an example.

For the lab instruction portion of the lab write-ups, the outer numbered items will be a high-level instruction for what needs to be done, for example:

1. Create a function to return if a number is a prime number AND positive

Below this task, there may be numerous sub-points that will help achieve the outer numbers item/task. For example:

   a. Create a local boolean variable
   b. Determine if the input argument is negative or positive
   c. Determine if the number is a prime number
   d. Return the boolean variable

The order these items are given corresponds to the order they will have with each other in the function. These sub-tasks may have a set of sub-points on their own. Consider "Create a local boolean variable" for sub-task "a":

      i. Name this variable 'isNumberPrimeANDPositive'
      ii. Initialize it to false

It is also possible for a sub-point to explain something and not require a course of action. These will be in *italic* font.
Altogether, you would have:

1. Create a function to return if a number is a prime number AND positive
   a. Create a local boolean variable
      i. Name this variable 'isNumberPrimeANDPositive'
      ii. Initialize it to false
   b. Determine if the input argument is negative or positive
      i. If the input argument is negative, return the boolean variable (which should be false)

ii. *Since the number is not positive, we can just return false and not execute the rest of the function because for the function to return true, the input number must be positive AND prime*

c. Determine if the number is prime using an external function
   i. Be sure to include the header where this function is defined
   ii. Capture the return value using the local boolean variable
      1. This means setting the variable equal to the function's return. For example:

```
booleanVar = externalFunction(inputNum);
```

   iii. *A lot of times, we do not like "reinventing the wheel." If there is already a function to achieve this concept, we can leverage it if it is okay to do so*

d. Return the boolean variable

The code would look like the following:

```
#include "SomeHeaderThatHasWhatWeNeed.h"
bool givenNumberIsPositiveAndPrime(int inputNum)
{
    bool isNumberPrimeANDPositive = false;
    if(inputNum < 0)
    {
       return isNumberPrimeANDPositive; // No need to continue, return now
    }
    isNumberPrimeAndPositive = externalFunction(inputNum);
    return isNumberPrimeAndPositive;
}
```

Of course this is a false function and may or may not do what we intend but this was created just to illustrate how the instructions correspond to the code. On a different note, this function can also be optimized can you find some areas of optimization? (You do not have to answer this).

NOTE - Sometimes, we will tell you precisely what to write. Sometimes, we will be extremely vague. This is by design. In industry, you will get tasks assigned to you by different people/teams, being able to move forward regardless of the amount of information received is a critical skill.

What will our program look like?

All of our labs will be written in C. We will have multiple layers of code that can be isolated into different "components."

- We will have low-level driver code that will interface with the hardware and be our "lowest level" code. We will do a lot of memory and register access within this code.
- We will have upper-level driver code that will act as an abstraction layer of the low-level driver code. Our LED driver and similar drivers will be considered "high-level drivers."

- We will have application code that is responsible for using all the other code (excluding low-level drivers and other components specified by the coding hierarchy) to add functionality to our code and solve the problem or task at hand.
- We will create and have components responsible for administrative stuff like scheduling which code/function should run and when.

We will be creating many file sets in this course: a source file and a header file.
The header files will contain prototypes and macros that clients (files that include the header file) can reference and use. The source files will contain the implementation of functions. The majority of code written will be in the source files

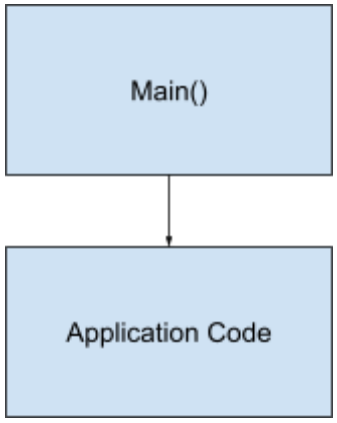
Coding Hierarchy and Coding Guidelines:

This course will enforce a coding hierarchy. Such a coding hierarchy aims to ensure “clean” code and enforce architectural constraints that a code base may have. It is entirely understandable for you not to know what that means. You will deal with many complex embedded systems in the industry with MANY components and firmware functionalities. You could have millions upon millions of lines of code. You do NOT want each file or component to include whatever files and headers they want. This is just bad practice. If you want a more in-depth explanation, there are many books on this concept, like Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series).

For this class, the coding hierarchy will be provided to you in each prelab. There will be an arrow pointing from one component to another this means that the component pointing to can include the header file of the component it is pointed at. For example, A points to B, A -> B. This means that in A’s header file, A can include B.h, also known as B’s header file. However, including A’s header file in B violates the coding hierarchy. Now there is something tricky, if B also points to C, that means it can include C.h in its header file, then A will also implicitly include C.h without having to explicitly state it. This is OK for this course, this is called transitive includes, and we aren’t going to spend time worrying about preventing that. If we wanted to avoid that, we would just include the header file of the other component in the SOURCE file(s). Confusing? Don’t stress too much just make sure to include the header of the file that the component directly points to.
A quick reference about C includes and source files:
https://onestepcode.com/include-several-files-c/

The guidelines are to verify your code is readable and understandable. Coding guidelines and format requirements are used all over the industry, so it doesn’t hurt to get exposed to these things now. With that being said, we are going to expose you to some coding hierarchies and guidelines.

Coding Hierarchy for Lab1:



Note - Files can only directly include files (the header files) to the components they “point” to in the diagram. So main.c (a source file in this case due to lack of header) will have:

```
#include ApplicationCode.h
```