CMPUT 274/5 - Tangible Computing **Fall 2019**
Weekly Exercise #6

Serial Counter

This exercise uses the same wiring for LEDs as the previous exercise. But it does not use any buttons.

Important Notes: Read This!

1. Your code must be in a file called `serial_counter.cpp` file that is compiled with the Makefile from the terminal. It must contain a `main()` function.

2. No template will be provided for this exercise.

3. As in the previous exercise, use pins 9,10,11,12,13 for LEDs. The "first LED" should be the one connected to pin 9. That is, you should think of these LEDs as corresponding to a 5-bit binary number with the LED on pin 9 corresponding to bit position 0.

   As before, marks will be deducted if you change the pins used for the lights. Your grading TA expects not to have to change the wiring for every submission! Please be kind to them by following these specifications.

In this exercise, you will maintain `uint8_t total`, a local variable in the `main()` function that is initially 0. The program will read characters from the serial monitor. Every time it reads a character corresponding to nonzero hex digit (`'0'` through `'F'`), the variable `total` will be incremented by the same amount (eg. if you press 7 then `total` goes up by 7). Every time you press the spacebar or the return/enter key, the variable `total` is reset back to 0. Any other character is ignored (i.e. has no effect on `total`).

Throughout, the 5 LEDs should always be illuminated to store the 5 **least significant** bits of `total`. That is, the LED on pin 9 should be illuminated if and only if the bit at position $i = 0$ of `total` is 1. The LED on pin 10 should be illuminated if and only if the bit at position $i = 1$ of `total` is 1. And so on (eg. pin 13 corresponds to bit position $i = 4$).

You must include two functions:

- `int8_t getHexValue(char digit)`
  Returns the value of the hexadecimal digit `digit` (recalling we use "digits" `A` through `F` to represent values 10 through 15). It should recognize both lowercase and uppercase characters for the hex digits from 10 to 15 (eg. `c` and `C` should both be recognized as digit 12).

  If `digit` is not a hexadecimal digit, this should just return -1. For full credit, you should **not** have 16 different checks like

```
if (digit == '0') {
 return 0;
```

```
}
else if (digit == '1') {
 return 1;
}
else if (digit == '2') {
 return 2;
}
... and so on, this is too messy
```

Use inequalities and arithmetic with ASCII values to check if `digit` is a valid hexadecimal digit and, if so, return the integer corresponding to its value. Do **not** use any built in functions that convert strings to integers, the point is to make you do the conversion yourself. You may still have a few cases to consider (eg. if `digit` is from `0` through `9`, or from `a` through `f`, or `A` through `F`).

- `void updateLEDs(uint8_t total)`
  Updates the 5 LEDs to illuminate the 5 least-significant bits of `total`, as per the description above. For full credit, you must use a loop to go over the 5 bit positions and you can only use bitwise operations in the body of the loop to determine if the corresponding LED should be illuminated. In particular, the `%` (mod) or `/` (division) operators cannot be used.

Further Comments
The only global variables you can use for this exercise are constant variables corresponding to pins. You cannot use any arrays except a global constant array to store the LED pin numbers.

You are permitted to print some information to the Serial monitor if you would like, but it is not required. For example, you might want to print the current value of `total` each time a key is pressed. That is acceptable. Just make sure the code is not messy with debug printing statements: keep your final submission clean.

It is also ok if the variable `total` overflows. You do not have to check that adding a value to it will cause it to exceed $2^8 - 1$.

Thought question (for your musing, not for submission): why don't you think we care about overflow with this problem?

Submission Guidelines:

Submit all of the following files in a .tar.gz called **serial_counter.tar.gz** or a .zip called **serial_counter.zip**:

- serial_counter.cpp, containing your implementation of the weekly exercise.
- the Makefile (introduced in class and available on the VM)
- your `README`, following the Code Submission Guidelines for Arduino projects

Note that your files must be named **exactly** as specified above.

Style
Your submission must use good style. Generally speaking, good style rules from Python that are applicable to C++ still apply: comments must be placed appropriately, variable names must make sense, you should exercise good use of functions, etc. With C++, you also **must have proper indenting**. While a C++ program with bad indenting can still be compiled, you must indent properly.

Finally, your program must have a `main()` function that calls a `setup()` function to initialize things like the Arduino, the pin modes, and anything else that is appropriate here. Note, Serial communication is not required for this weekly exercise but you will not be penalized for initializing it anyway.