CMSC 216 Arduino Exercise #2 Summer 2019

Pixel Spin **Due: Monday, Jul 29, 2019, 11:30PM**

1 Objectives

To practice basic Arduino development, ensure your environment is set up with the TwoSixteen library, practice processing input from a push button, develop a debugging strategy for the Arduino device, and practice use of arrays and (optionally) structs.

We will use init_neopixel_blit and neopixel_blit, init_serial_stdio and printf, as well as functions you've already used (pinMode, digitalRead, and delay) while writing setup and loop.

2 Overview

Your task is to make the neopixels (RGB LEDs) "spin", faster when the right button (pin 19) is pressed, slower when the left button (pin 4) is pressed. The following video illustrates the functionality you are expected to implement:

https://tinyurl.com/y9jbdvjd

To save joules and your eyesight, do not use full brightness. (A color value of 30 is plenty.)

You must include (and use) the TwoSixteen library.

Submit by uploading a single source file to submit.cs. The single source file may be named with the .ino extension from the Arduino IDE or a .c extension if developing using an alternate method with an arduino makefile.

Unlike other assignments for this course, you can work together with other classmates, but like all assignments, you may NOT share, exchange, post, etc., any code.

3 TwoSixteen library

3.1 Installation

You may have seen instructions for installing TwoSixteen. Download it from:

http://www.cs.umd.edu/~nelson/classes/resources/circuitplayground/TwoSixteen-0.0.5.zip

To install, from within the IDE, in the "Sketch" menu, "Include Library", "Add .ZIP Library".

3.2 Use of stdio

The `serial_interface.h` file declares init_serial_stdio(). Call that function in setup(). Call printf() as you like, but be careful calling it too frequently as the capture buffer on the submit server is finite and it can take time on the hardware to complete the printf.

3.3 Use of neopixels

The `neopixel_blit.h` file declares init_neopixel_blit and neopixel_blit. Its contents are below.

```
#include <stdint.h>

typedef uint8_t Pixels[10][3];
typedef struct Pixel {
  uint8_t green, red, blue;
} PixelStruct[10];

#define PIX_GRN 0
#define PIX_RED 1
#define PIX_BLU 2
```

```
#ifdef __cplusplus
extern "C" {
#endif

  void init_neopixel_blit();
  void neopixel_blit(const void *pixels); /* typically, Pixels or PixelStruct */

#ifdef __cplusplus
}
#endif
```

To use this, call the init_neopixel_blit function in setup() to ensure that the lights go out and possibly to initialize. Declare an instance of Pixels, or of PixelStruct, to pass to neopixel_blit. This function will send the colors of all pixels. The only way to change the color of a single pixel is to set the color of all pixels, typically to a modified copy you've kept.

Set a color by setting your defined pixels as you like. For example,

```
Pixels pix = {{0}};
pix[1][PIX_BLU] = 20;
neopixel_blit(pix);
```

The colors passed to neopixel_blit() are in green-red-blue order, not RGB as you might be familiar with, but you won't even notice if you use the constants as array subscripts or use the structure.

The uint8_t type is an unsigned integer that occupies 8 bits, meaning it can take a value from 0-255.

When compiled from within a C++ compiler, the symbol __cplusplus is defined, which wraps the function prototypes in a block that tells the C++ compiler that the functions are to be called as C functions.

4 Specifications

1. Exactly one neopixel may be active at a time. Don't submit a comet-like pattern where other pixels stay active. Tests will be looking for whether the pixel previously lit is turned off.
2. Your pixel may be any color you choose; I recommend color values at most 30.
3. You must printf("delay: %dms\n", d); once per revolution (cycle). (Name our variable as you like, just print the delay.)
4. Start with a delay between frames of 100ms, such that the cycle completes about once per second.
5. Check button state once per frame, not just once per cycle.
6. Your implementation should accelerate the spin when the button is pushed, but not continue to accelerate while the button remains pushed. Ensure the button is released before allowing a button push to accelerate again.
7. You may assume the button remains pressed through any delay. You may, but do not need to create a helper function that polls the button every, say, 100ms while trying to wait for 1 second.
8. Pushing the left button should slow the spin, doubling delay up to 1 second delay per frame. (If you double to compute a delay greater than 1s, set it to 1s.)
9. Pushing the right button should speed up the spin, halving delay down to 2 milliseconds delay per frame. (If you halve to compute a delay less than 2ms, set it to 2ms.)
10. Take care that delay does not become negative. If the delay becomes terribly large, you may need to double-press the reset button to reprogram.

11. It takes about 300μs to send colors to the pixels, though you need not account for this time. (That is, it's fine to delay for $d$ms directly, not $d - 0.3$ ms.

12. You may use the switch to alter how the pixels spin, as long as both switch positions are valid: you may change the color or direction.

13. You must not use the CircuitPlayground library.

5 Project Requirements

1. Your grade will be based on the results obtained from the submit server. It is your responsibility to verify that your program generates the expected results in the submit server.

2. Your program should be written using good programming style as defined at

   http://www.cs.umd.edu/~nelson/classes/resources/cstyleguide/

   Please define symbolic constants for left and right button pins. (In contrast to the last Arduino exercise that relaxed that requirement.)

3. Functionality should be distributed to the setup and loop functions as expected: one-time initialization code happens in setup, repeated testing of the switch happens in loop. Your code must at least occasionally return from loop.

4. Do not use the CircuitPlayground.redLED() or CircuitPlayground.slideSwitch() functions or other components of the circuit playground library. The CircuitPlayground.begin() function in particular can cause tests to fail.

5. Ensure that you are not taking advantage of the Arduino IDE's ability to infer which include files are necessary and automatically add them. That is, make sure your sketch explicitly includes Arduino.h (Capitalized), neopixel_blit.h, serial_interface.h, and stdio.h. If you notice that your sketch includes two different files in a single #include line, this is not supported, split them.

6. Do not blank the neopixels at any time after initialization. Each time neopixel_blit is called, it must illuminate exactly one LED.

7. We expect you to explicitly set pinMode for the input buttons. Technically, setting pinMode to input is redundant; this to encourage a general "initialize your variables" practice. You do not need to set pinMode for the neopixel pin (17), since this is handled by init_neopixel_blit.

8. The change in speed should happen before releasing the button, but not again until being released.

9. Make sure you submit often and check tests results. Your code might seem to work when run in the device, but due to the simulator used by the submit server, you might not be passing tests. You have to submit and check submit server results often so you can see a TA early in case you are not passing tests.

10. Do not clear (turn off) all neopixels at any time after initialization. If you need to clear a pixel, just set that particular pixel to 0 and not all of them (do not set to 0 something that is already 0). You will not pass submit server tests if you don't follow this rule (even though the program seems to be working in your device).

11. Everything you need to complete this assignment is in the examples we have provided. Visit:

    http://www.cs.umd.edu/~nelson/classes/resources/circuitplayground/basics.shtml

6 Submitting your assignment

1. Visit submit.cs.umd.edu and upload the source file. You may submit a zip file, but note that it must not place your source file in a subdirectory, and if it includes additional c files, the submit server will likely choose to compile all of them together and the build will fail.

2. Your assignment must be electronically submitted by the date and time above to avoid losing credit. See the course syllabus for details.

3. Make sure you check your release test results.

7 Grading Criteria

Your assignment grade will be determined with the following weights:

Results of public tests	84%
Results of release tests	16%

8 Academic integrity statement

Please **carefully read** the academic integrity section of the course syllabus. **Any evidence** of impermissible cooperation on assignments, use of disallowed materials or resources, or unauthorized use of computer accounts, **will be submitted** to the Student Honor Council, which could result in an XF for the course, or suspension or expulsion from the University. Be sure you understand what you are and what you are not permitted to do with regard to academic integrity when it comes to assignments. These policies apply to all students, and the Student Honor Council does not consider lack of knowledge of the policies to be a defense for violating them. Full information is found in the course syllabus – please review it at this time.