ECE 177 Lab 10 Spring 2019

Purpose: More practice with arrays, using pointers and pointer arithmetic. Using an ARV timer.

Assignment: For this lab you will rewrite the previous lab exercise using pointers. (Note that you must first get the previous lab to work if you haven't previously done so!) For the most part you will simply convert each array indexing operation (using "**`str[i]`**") to a pointer operation (using "**`*p_str`**"). Passing arrays to functions will be done by passing a pointer to the first element of the array. Effectively this is the same as before, but the syntax is different in the function prototype. The call to the function will be unchanged. There should not be a single square bracket ('[' or ']') in your program other than in array declarations (**`mapkey()`** can be unchanged from the previous lab). Use pointers "properly" – do not simply change x[i] to *(x+i). Also, do not use any "counters"; i.e.,, don't say, for example, "`for (i=0; i<4; i++) ...`" Instead, use your pointers to keep track.

You will also write some functions that allow you to delay more precisely:

`delaycycles()` will delay (fairly) precisely a number of cycles (microseconds) passed to it as an argument using the AVR's "`timer 1`". This timer is essentially a "volatile" 16-bit value in memory called "**`TCNT1`**" that auto-magically counts up once per microsecond. This timer is enabled in the starter code by writing a "0x01" to the "**`TCCR1B`**" register. Write a value to **`TCNT1`** and it continues counting from there. When the counter "rolls over" from 0xFFFF to 0x0000 it auto-magically sets a flag called "**`TOV1`**" in a register called "**`TIFR1`**" ("**`TOV1`**" represents the bit number – see class notes).

So, **`delaycycles()`** will function as follows: first subtract some "special" number from the number of cycles to delay, to account for "overhead". (We'll figure out the special number.) Write the negative of this result to **`TCNT1`**, clear the **`TOV1`** bit of **`TIFR1`** by writing a "1" to this bit (weird, but see class notes) and then wait for **`TOV1`** to go to 1 and return. (The function contains a total of 3-4 lines of code.)

`mydelayms()` will be rewritten to include a single `for` loop. The *body* of the loop will call `delaycycles()` to delay exactly 1000 cycles (one millisecond).

Prelab: Rewrite the three CodeLab exercises using pointers. The set of Lab 9 exercises is duplicated in a "Lab 10" section of CodeLab. Note that while simply resubmitting your Lab 9 solutions may give a correct answer in CodeLab, I will check your solution by hand to make sure it has been properly rewritten.

Notes:

The above will get you a B grade. For an A, have your code play tones using the speaker: play a single friendly beep if OPEN LOCK is displayed and four error beeps if the password is entered incorrectly. Hook the speaker to PORTC pin 1 (same as leftmost LED) – move the speaker if it was elsewhere before. To play a tone of a certain duration, use a "`for`" loop whose body simply toggles the speaker and then delays using `delaycycles()`. Delay according to the tone frequency (period is 1/f, so delay half that after each toggle). The `for` loop should loop a number of times determined by the desired duration and the above period. See class notes for further information on how to accomplish this.