

Purpose: Using arrays. More programming practice.

**Assignment:** For this lab you will use the "hex" keypad and write a "password entry" program. The program will wait for a user to enter a password on the keypad and will then display either "Open Lock" for two seconds if correct or it will flash "Error" 4 times if incorrect. It will then loop back and wait for another password entry. Your password will be stored in an array inside your program.

**Notes:** Your program should use five functions: `getKey()`, `mapkey()`, `getpw()`, `compareArrays()` and `mydelaysms()`. See the prototypes for these functions in the starter code.

Use your `getKey()` and `mapkey()` verbatim from the previous lab – no changes.

Function `getpw()` will record keys pressed on the keypad (as returned by "mapkey()") until a '#' key is hit (much like the previous lab). As before, '\*' will be used as a "backspace" key. The keys pressed (entered password) are stored in an array (buffer) that is passed to the routine as the second argument. The first argument to the function is the maximum size of the array. Any digit or letter keys hit when the buffer is full will be ignored. The function's return value will be the number of values in the buffer. The function can operate as follows: In a loop, wait for all keys to be released, wait for a key to be pressed, call `mapkey()` with that key. If the mapped key is a digit or letter (value 0 to 13), then add it to the buffer if the buffer is not full (ignore it otherwise). If the mapped key is an '\*' then reduce the current size of the buffer by one (but don't make it negative size). If the mapped key is a '#' then return the size of the buffer. Hint: the structure of this routine is very similar to "getnum()" from the previous lab, but rather than modifying `result` when a key is pressed, we add it to the buffer.

Function `compareArrays()` will compare two arrays and will return 1 if the arrays are equal, and 0 otherwise. In addition to the two arrays passed as arguments, a third argument will be the size of the arrays. The function should also work correctly if the size is given as zero.

Function `mydelaysms()` will simply delay the given number of milliseconds. For this routine simply call `HAL_Delay()` with the value. We'll modify the function next week to do our own delay. Any of your delays will use `mydelaysms()`.

Your `main()` function will include (as local variables) an array which stores the correct password and a variable that indicates the password length. Your code should be written in such a way that changing this length and the array is all that is needed to make a corresponding change in the password. Your main function will also include an array (buffer) to hold the password entered by the user (size is `BUFSIZE`, see starter code). Your main code will (in an infinite loop): call `getpw()` to get a password entered by the user. Then, if the password is the correct length (check the return value of `getpw()`) and if the password is correct (call `compareArrays()`) will then display "Open lock" for two seconds. If it is incorrect, then blink "Error" four times in the display over a two second period.

Doing the above will get you a "C" grade. For an "B" have your `getpw()` light one LED (starting at the right) for every value in the buffer. Of course, remove one lit LED when the "Backspace" key ('\*') is hit. For the "A" grade have your main program change it's behavior such that after each correct

password entry, it will display “Enter password”, then call `getpw()` again, then change the saved password to be the password just returned, then blank the display. If an incorrect password is entered, flash “Error” four times as before.

**Prelab:** CodeLab problems related to this lab must be completed before coming to lab.