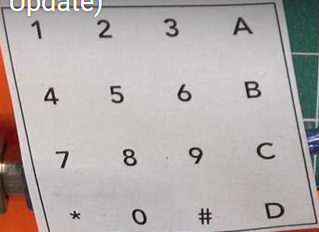
**Day 15 Notes**

Example Circuit

My Circuit



Preliminary Thoughts:

* Change Code to only give valid or not valid after all PIN characters are entered as suggested in the Day 15 Lesson video.
* Add required entry of a NEW PIN at startup, before entering Loop.
* Require a second verifying entry of the PIN before completing a change.
* Allow the PIN to be from 4 to 8 characters long. Use the “#” key to end entry of a new PIN.
* Can we make user entered Data (New PIN) persist past power off / on cycle? If so, keep the New Pin after restart and don’t require another NEW PIN on restart.

Thank you for catching that! Let’s update the code to address both requirements:

1. \*\*User-Defined Starting PIN\*\*: Prompt the user to set up a new PIN from the keypad at the start, rather than using a hard-coded PIN.

2. \*\*Variable PIN Length\*\*: Allow the PIN to range from 4 to 8 characters, finalized by the `#` key.

Here's the revised code:

```cpp

// Define maximum and minimum PIN length

const int minPinLength = 4;

const int maxPinLength = 8;

const int maxAttempts = 5;

char currentPIN[maxPinLength + 1]; // Stores the active PIN

int attemptCount = 0; // Track number of attempts

void setup() {

Serial.begin(9600); // Start serial communication

pinMode(LED\_BUILTIN, OUTPUT); // Set up onboard LED for feedback

// Prompt for initial user-defined PIN setup

Serial.println("Set a new PIN (4-8 characters). End with #:");

setupNewPIN();

}

void loop() {

if (attemptCount >= maxAttempts) {

Serial.println("Too many failed attempts. Access locked.");

while (true); // Lock the system

}

Serial.println("Enter PIN:");

if (validatePIN()) {

Serial.println("Access Granted!");

digitalWrite(LED\_BUILTIN, HIGH); // Turn on LED for valid entry

delay(1000); // Hold success for visibility

} else {

attemptCount++;

Serial.print("Incorrect PIN. Attempts remaining: ");

Serial.println(maxAttempts - attemptCount);

digitalWrite(LED\_BUILTIN, LOW); // Turn off LED for invalid entry

delay(1000);

}

}

// Set up a new PIN with verification

void setupNewPIN() {

char tempPIN[maxPinLength + 1]; // Temporary storage for first PIN entry

// First entry of new PIN

Serial.println("Enter new PIN:");

if (getPINInput(tempPIN)) {

Serial.println("Re-enter new PIN for verification:");

// Second entry to confirm

char verifyPIN[maxPinLength + 1];

if (getPINInput(verifyPIN) && strcmp(tempPIN, verifyPIN) == 0) {

strcpy(currentPIN, tempPIN); // Set the new PIN

Serial.println("New PIN set successfully!");

} else {

Serial.println("PINs did not match. Restart setup.");

setupNewPIN(); // Retry setup if verification fails

}

}

}

// Get PIN input with "#" to end entry

bool getPINInput(char\* pinBuffer) {

int charIndex = 0;

while (true) {

if (Serial.available()) {

char inputChar = Serial.read();

// Check for end of entry with "#"

if (inputChar == '#') {

pinBuffer[charIndex] = '\0'; // End the PIN string

if (charIndex >= minPinLength && charIndex <= maxPinLength) {

return true;

} else {

Serial.println("Invalid PIN length. Try again.");

return false; // Invalid length

}

}

// Add character if within limits

if (charIndex < maxPinLength) {

pinBuffer[charIndex] = inputChar;

Serial.print("\*"); // Mask input with \*

charIndex++;

}

}

}

}

// Validate entered PIN against currentPIN

bool validatePIN() {

char enteredPIN[maxPinLength + 1];

if (getPINInput(enteredPIN)) {

return strcmp(enteredPIN, currentPIN) == 0;

}

return false;

}

```

### Key Changes:

- \*\*User-defined Starting PIN\*\*: The `setupNewPIN()` function now requires a user-defined PIN at startup. This process involves two entries—the first for creating the PIN and the second for verification.

- \*\*Variable PIN Length\*\*: `getPINInput()` checks that the entered PIN meets the 4–8 character requirement. The `#` key finalizes the input, making it possible to vary the PIN length.

### How It Works:

- \*\*Set a New PIN\*\*: At startup, `setupNewPIN()` is called to prompt the user to enter and confirm a new PIN.

- \*\*Variable Length\*\*: The user-defined PIN can be between 4 and 8 characters, and the `#` key signifies the end of input.

- \*\*Attempt Limit\*\*: The program limits attempts to 5. After 5 incorrect attempts, the system locks.

This should now fully meet your requirements for a customizable PIN setup and validation process! Let me know if you need further customization or if you’d like to add any new features.

Thank you for sharing the full code! I’ll integrate the requested updates into this base while preserving the existing RGB LED, buzzer functionality, and new pin locations.

### Key Changes

1. \*\*Variable PIN Length\*\*: Allow a PIN from 4 to 8 characters using `#` to end the input.

2. \*\*User-Defined PIN at Startup\*\*: Require the user to set up a new PIN and confirm it on startup.

3. \*\*Limited Attempts\*\*: Limit the number of PIN entry attempts to 5.

Here’s the revised code with these updates:

```cpp

#include "Arduino.h"

#include <Keypad.h>

// Constants for the minimum and maximum PIN length

const int minPinLength = 4;

const int maxPinLength = 8;

const int maxAttempts = 5;

// Define what characters will be returned by each button

const char BUTTONS[4][4] = {

{ '1', '2', '3', 'A' },

{ '4', '5', '6', 'B' },

{ '7', '8', '9', 'C' },

{ '\*', '0', '#', 'D' }

};

const byte ROW\_PINS[4] = { 5, 4, 3, 2 };

const byte COL\_PINS[4] = { 6, 7, 8, 13 };

Keypad heroKeypad = Keypad(makeKeymap(BUTTONS), ROW\_PINS, COL\_PINS, 4, 4);

const byte BUZZER\_PIN = 12;

const byte RED\_PIN = 11;

const byte GREEN\_PIN = 10;

const byte BLUE\_PIN = 9;

char currentPIN[maxPinLength + 1];

int attemptCount = 0;

void setup() {

pinMode(RED\_PIN, OUTPUT);

pinMode(GREEN\_PIN, OUTPUT);

pinMode(BLUE\_PIN, OUTPUT);

displayColor(128, 0, 0);

Serial.begin(9600);

Serial.println("Set a new PIN (4-8 characters). End with #:");

setupNewPIN();

}

void loop() {

if (attemptCount >= maxAttempts) {

Serial.println("Too many failed attempts. Access locked.");

while (true);

}

Serial.println("Enter PIN:");

if (validatePIN()) {

Serial.println("Access Granted!");

giveSuccessFeedback();

attemptCount = 0;

} else {

attemptCount++;

Serial.print("Incorrect PIN. Attempts remaining: ");

Serial.println(maxAttempts - attemptCount);

giveErrorFeedback();

}

}

void setupNewPIN() {

char tempPIN[maxPinLength + 1];

Serial.println("Enter new PIN:");

if (getPINInput(tempPIN)) {

Serial.println("Re-enter new PIN for verification:");

char verifyPIN[maxPinLength + 1];

if (getPINInput(verifyPIN) && strcmp(tempPIN, verifyPIN) == 0) {

strcpy(currentPIN, tempPIN);

Serial.println("New PIN set successfully!");

giveSuccessFeedback();

} else {

Serial.println("PINs did not match. Restart setup.");

setupNewPIN();

}

}

}

// Get PIN input with "#" to end entry

bool getPINInput(char\* pinBuffer) {

int charIndex = 0;

while (true) {

char key = heroKeypad.getKey();

if (key) {

if (key == '#') {

pinBuffer[charIndex] = '\0';

if (charIndex >= minPinLength && charIndex <= maxPinLength) {

return true;

} else {

Serial.println("Invalid PIN length. Try again.");

return false;

}

}

if (charIndex < maxPinLength) {

pinBuffer[charIndex] = key;

Serial.print("\*");

charIndex++;

}

giveInputFeedback();

}

}

}

bool validatePIN() {

char enteredPIN[maxPinLength + 1];

if (getPINInput(enteredPIN)) {

if (strcmp(enteredPIN, currentPIN) == 0) {

giveSuccessFeedback();

return true;

}

}

giveErrorFeedback();

return false;

}

void displayColor(byte red\_intensity, byte green\_intensity, byte blue\_intensity) {

analogWrite(RED\_PIN, red\_intensity);

analogWrite(GREEN\_PIN, green\_intensity);

analogWrite(BLUE\_PIN, blue\_intensity);

}

void giveInputFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 880, 200);

delay(200);

displayColor(0, 0, 128);

}

void giveSuccessFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 300, 200);

delay(200);

tone(BUZZER\_PIN, 500, 500);

delay(500);

displayColor(0, 128, 0);

}

void giveErrorFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 300, 200);

delay(200);

tone(BUZZER\_PIN, 200, 500);

delay(500);

displayColor(128, 0, 0);

}

```

### Key Updates:

1. \*\*User-Defined Starting PIN\*\*: At startup, the user is prompted to enter a new PIN, followed by a verification entry.

2. \*\*Variable PIN Length\*\*: The user-defined PIN can be 4-8 characters, finalized with `#`.

3. \*\*Limited Attempts\*\*: The user has up to 5 attempts to enter the correct PIN, after which the system locks.

### How It Works:

- \*\*New PIN Setup\*\*: `setupNewPIN()` gathers and verifies a new PIN from the user.

- \*\*PIN Entry and Validation\*\*: `validatePIN()` checks the entered PIN against `currentPIN`.

- \*\*Feedback Functions\*\*: The `giveInputFeedback()`, `giveSuccessFeedback()`, and `giveErrorFeedback()` functions handle the RGB LED and buzzer responses.

With this code, you’ll have a dynamic, user-defined PIN system with clear feedback and limited attempts. Let me know if there’s anything more to add or tweak!

Let’s adjust the code to make `#` a special character that ends the PIN input but isn’t included in the PIN itself. Additionally, we’ll ensure `\*` is recognized as the prompt for entering a new PIN without including it in the PIN entry.

### Revised Code

Below are the updates addressing both issues:

1. \*\*Ignore `#` in the PIN Array\*\*: When the user presses `#`, the entry ends without adding `#` to the PIN.

2. \*\*Allow `\*` for PIN Change\*\*: The `\*` character will be handled as a command to initiate a new PIN setup without affecting PIN entry.

Here’s the modified code with these changes:

```cpp

#include "Arduino.h"

#include <Keypad.h>

// Constants for the minimum and maximum PIN length

const int minPinLength = 4;

const int maxPinLength = 8;

const int maxAttempts = 5;

// Define what characters will be returned by each button

const char BUTTONS[4][4] = {

{ '1', '2', '3', 'A' },

{ '4', '5', '6', 'B' },

{ '7', '8', '9', 'C' },

{ '\*', '0', '#', 'D' }

};

const byte ROW\_PINS[4] = { 5, 4, 3, 2 };

const byte COL\_PINS[4] = { 6, 7, 8, 13 };

Keypad heroKeypad = Keypad(makeKeymap(BUTTONS), ROW\_PINS, COL\_PINS, 4, 4);

const byte BUZZER\_PIN = 12;

const byte RED\_PIN = 11;

const byte GREEN\_PIN = 10;

const byte BLUE\_PIN = 9;

char currentPIN[maxPinLength + 1];

int attemptCount = 0;

void setup() {

pinMode(RED\_PIN, OUTPUT);

pinMode(GREEN\_PIN, OUTPUT);

pinMode(BLUE\_PIN, OUTPUT);

displayColor(128, 0, 0);

Serial.begin(9600);

Serial.println("Set a new PIN (4-8 characters). End with #:");

setupNewPIN();

}

void loop() {

char button\_character = heroKeypad.getKey();

if (button\_character == '\*') { // Trigger new PIN setup

giveInputFeedback();

Serial.println("Access current PIN to set new PIN:");

if (validatePIN()) {

setupNewPIN();

} else {

Serial.println("Access Denied. Incorrect PIN.");

}

} else if (button\_character == '#') { // Trigger PIN validation for access

giveInputFeedback();

if (validatePIN()) {

Serial.println("Access Granted!");

giveSuccessFeedback();

attemptCount = 0;

} else {

attemptCount++;

Serial.print("Incorrect PIN. Attempts remaining: ");

Serial.println(maxAttempts - attemptCount);

giveErrorFeedback();

if (attemptCount >= maxAttempts) {

Serial.println("Too many failed attempts. Access locked.");

while (true); // Lock system after too many attempts

}

}

}

}

// Set up a new PIN with verification

void setupNewPIN() {

char tempPIN[maxPinLength + 1];

Serial.println("Enter new PIN:");

if (getPINInput(tempPIN)) {

Serial.println("Re-enter new PIN for verification:");

char verifyPIN[maxPinLength + 1];

if (getPINInput(verifyPIN) && strcmp(tempPIN, verifyPIN) == 0) {

strcpy(currentPIN, tempPIN);

Serial.println("New PIN set successfully!");

giveSuccessFeedback();

} else {

Serial.println("PINs did not match. Restart setup.");

setupNewPIN();

}

}

}

// Get PIN input with "#" to end entry

bool getPINInput(char\* pinBuffer) {

int charIndex = 0;

while (true) {

char key = heroKeypad.getKey();

if (key) {

if (key == '#') { // End of PIN entry

pinBuffer[charIndex] = '\0';

if (charIndex >= minPinLength && charIndex <= maxPinLength) {

return true;

} else {

Serial.println("Invalid PIN length. Try again.");

return false;

}

}

// Add character if within limits and ignore '\*'

if (charIndex < maxPinLength && key != '\*') {

pinBuffer[charIndex] = key;

Serial.print("\*");

charIndex++;

}

giveInputFeedback();

}

}

}

// Validate entered PIN against currentPIN

bool validatePIN() {

char enteredPIN[maxPinLength + 1];

if (getPINInput(enteredPIN)) {

if (strcmp(enteredPIN, currentPIN) == 0) {

giveSuccessFeedback();

return true;

}

}

giveErrorFeedback();

return false;

}

void displayColor(byte red\_intensity, byte green\_intensity, byte blue\_intensity) {

analogWrite(RED\_PIN, red\_intensity);

analogWrite(GREEN\_PIN, green\_intensity);

analogWrite(BLUE\_PIN, blue\_intensity);

}

void giveInputFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 880, 200);

delay(200);

displayColor(0, 0, 128);

}

void giveSuccessFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 300, 200);

delay(200);

tone(BUZZER\_PIN, 500, 500);

delay(500);

displayColor(0, 128, 0);

}

void giveErrorFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 300, 200);

delay(200);

tone(BUZZER\_PIN, 200, 500);

delay(500);

displayColor(128, 0, 0);

}

```

### Explanation of Key Updates

1. \*\*`#` as a Special Character\*\*:

- In `getPINInput()`, `#` is used to indicate the end of the PIN entry and is \*\*not\*\* included in `pinBuffer`. When the user presses `#`, it checks if the PIN length is within the allowed range (4-8 characters) and finalizes the input.

2. \*\*`\*` to Trigger PIN Change\*\*:

- In `loop()`, pressing `\*` prompts the user to enter the current PIN to gain access to change the PIN. This special character is ignored in the actual PIN entry.

These changes should now allow for flexible PIN entry, with `#` as an end character and `\*` to start the PIN change process, without including them in the actual PIN content. Let me know how this version works for you!

++++++++++++++++++++++++++++++++++++++++++++++++++++++

It looks like the functionality is OK for PIN entry.

In the case of a new pin enter 4 to 8 characters followed by “#” character. The characters before the “#” become the PIN.

In the case of verifying an entered PIN enter 4 to 8 characters followed by the “#” character. If the entered characters match the set PIN validation succeeds.

Please update the code to reflect the following performance options:

If locked the only option is to unlock by entering the correct PIN. Make sure the prompts are in place.

If unlocked the options are to lock by entering the “#” character, change the PIN by entering the “\*” character, or work by entering any other character (just echo the character to serial and keep the LED green). Prompt, print, and set LED and buzzer accordingly.

Here’s an updated version of the code that adds a locked/unlocked state and provides options based on the current state. When the system is \*\*locked\*\*, only a correct PIN entry will unlock it. When \*\*unlocked\*\*, users can:

- Enter `#` to lock the system.

- Enter `\*` to change the PIN.

- Enter any other character to “work,” which will be echoed to the Serial Monitor.

### Updated Code

```cpp

#include "Arduino.h"

#include <Keypad.h>

// Constants for the minimum and maximum PIN length

const int minPinLength = 4;

const int maxPinLength = 8;

const int maxAttempts = 5;

char currentPIN[maxPinLength + 1]; // Stores the active PIN

int attemptCount = 0; // Track number of attempts

bool isLocked = true; // Track lock state

const char BUTTONS[4][4] = {

{ '1', '2', '3', 'A' },

{ '4', '5', '6', 'B' },

{ '7', '8', '9', 'C' },

{ '\*', '0', '#', 'D' }

};

const byte ROW\_PINS[4] = { 5, 4, 3, 2 };

const byte COL\_PINS[4] = { 6, 7, 8, 13 };

Keypad heroKeypad = Keypad(makeKeymap(BUTTONS), ROW\_PINS, COL\_PINS, 4, 4);

const byte BUZZER\_PIN = 12;

const byte RED\_PIN = 11;

const byte GREEN\_PIN = 10;

const byte BLUE\_PIN = 9;

void setup() {

pinMode(RED\_PIN, OUTPUT);

pinMode(GREEN\_PIN, OUTPUT);

pinMode(BLUE\_PIN, OUTPUT);

Serial.begin(9600);

Serial.println("Set a new PIN (4-8 characters). End with #:");

setupNewPIN();

displayColor(128, 0, 0); // Red LED indicates locked state

}

void loop() {

if (isLocked) {

Serial.println("System is locked. Enter PIN to unlock:");

if (validatePIN()) {

isLocked = false;

Serial.println("System Unlocked! Options: '#' to lock, '\*' to change PIN, any other key to work.");

displayColor(0, 128, 0); // Green LED indicates unlocked state

} else {

attemptCount++;

if (attemptCount >= maxAttempts) {

Serial.println("Too many failed attempts. Access locked permanently.");

while (true); // Lock indefinitely

}

}

} else {

char button\_character = heroKeypad.getKey();

if (button\_character) {

if (button\_character == '#') { // Lock the system

isLocked = true;

Serial.println("System locked. Enter PIN to unlock.");

displayColor(128, 0, 0); // Red LED for locked state

} else if (button\_character == '\*') { // Change PIN

Serial.println("Enter current PIN to change it:");

if (validatePIN()) {

Serial.println("PIN accepted. Setting up new PIN.");

displayColor(128, 80, 0); // Yellow LED while changing PIN

setupNewPIN();

Serial.println("New PIN set successfully. System unlocked.");

displayColor(0, 128, 0); // Green LED for unlocked state

} else {

Serial.println("Incorrect PIN. Returning to unlocked mode.");

displayColor(0, 128, 0); // Keep green LED for unlocked state

}

} else { // Any other character to "work"

Serial.print("Working... You pressed: ");

Serial.println(button\_character);

giveInputFeedback(); // Short beep and blue LED for each work action

displayColor(0, 128, 0); // Maintain green LED for unlocked state

}

}

}

}

void setupNewPIN() {

char tempPIN[maxPinLength + 1];

Serial.println("Enter new PIN:");

if (getPINInput(tempPIN)) {

Serial.println("Re-enter new PIN for verification:");

char verifyPIN[maxPinLength + 1];

if (getPINInput(verifyPIN) && strcmp(tempPIN, verifyPIN) == 0) {

strcpy(currentPIN, tempPIN);

Serial.println("New PIN set successfully!");

giveSuccessFeedback();

} else {

Serial.println("PINs did not match. Restarting setup.");

setupNewPIN();

}

}

}

bool getPINInput(char\* pinBuffer) {

int charIndex = 0;

while (true) {

char key = heroKeypad.getKey();

if (key) {

if (key == '#') {

pinBuffer[charIndex] = '\0';

if (charIndex >= minPinLength && charIndex <= maxPinLength) {

return true;

} else {

Serial.println("Invalid PIN length. Try again.");

return false;

}

}

if (charIndex < maxPinLength) {

pinBuffer[charIndex] = key;

Serial.print("\*");

charIndex++;

}

giveInputFeedback();

}

}

}

bool validatePIN() {

char enteredPIN[maxPinLength + 1];

if (getPINInput(enteredPIN)) {

if (strcmp(enteredPIN, currentPIN) == 0) {

giveSuccessFeedback();

return true;

}

}

giveErrorFeedback();

return false;

}

void displayColor(byte red\_intensity, byte green\_intensity, byte blue\_intensity) {

analogWrite(RED\_PIN, red\_intensity);

analogWrite(GREEN\_PIN, green\_intensity);

analogWrite(BLUE\_PIN, blue\_intensity);

}

void giveInputFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 880, 200);

delay(200);

displayColor(0, 0, 128);

}

void giveSuccessFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 300, 200);

delay(200);

tone(BUZZER\_PIN, 500, 500);

delay(500);

displayColor(0, 128, 0);

}

void giveErrorFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 300, 200);

delay(200);

tone(BUZZER\_PIN, 200, 500);

delay(500);

displayColor(128, 0, 0);

}

```

### Explanation of Changes

1. \*\*Lock/Unlock Functionality\*\*:

- The `isLocked` variable tracks the system’s state.

- When \*\*locked\*\*, only a correct PIN can unlock it, and all other actions are disabled.

- When \*\*unlocked\*\*, the user has three options:

- `#`: Lock the system.

- `\*`: Change the PIN (requires re-entering the current PIN for security).

- Any other character: Print the character to the Serial Monitor and simulate "working."

2. \*\*Color and Sound Feedback\*\*:

- \*\*Locked\*\*: Red LED indicates the system is locked.

- \*\*Unlocked\*\*: Green LED when the system is unlocked.

- \*\*Input Feedback\*\*: Short beep and blue LED on valid inputs.

- \*\*Success/Failure\*\*: Distinct success/failure feedback with LED colors and buzzer tones.

This setup should cover all requested functionality. Let me know if you’d like further customization or additional features!

=======================================

**Persistant PIN with EEPROM Library**

Can we make user entered Data (New PIN) persist past power off / on cycle? If so, keep the New Pin after restart and don’t require another NEW PIN on restart.

Yes, we can make the user-entered PIN persist across power cycles by saving it to the \*\*EEPROM\*\* memory on the HERO board. EEPROM (Electrically Erasable Programmable Read-Only Memory) is a non-volatile memory, meaning it retains data even when the board is powered off.

Here’s how to modify the code to save the PIN in EEPROM and retrieve it on startup:

1. \*\*Include the EEPROM Library\*\*: This library allows you to read from and write to the EEPROM.

2. \*\*Save the PIN\*\*: Write the PIN to EEPROM each time the user sets a new one.

3. \*\*Load the PIN on Startup\*\*: Read the stored PIN from EEPROM on startup. If no PIN is stored, the user will be prompted to set a new one.

### Code Example

Here’s the modified version of your code with EEPROM functionality:

```cpp

#include "Arduino.h"

#include <Keypad.h>

#include <EEPROM.h> // Include EEPROM library

// Constants for the minimum and maximum PIN length

const int minPinLength = 4;

const int maxPinLength = 8;

const int maxAttempts = 5;

const int eepromStartAddress = 0; // Starting address in EEPROM for PIN storage

char currentPIN[maxPinLength + 1]; // Stores the active PIN

int attemptCount = 0; // Track number of attempts

bool isLocked = true; // Track lock state

const char BUTTONS[4][4] = {

{ '1', '2', '3', 'A' },

{ '4', '5', '6', 'B' },

{ '7', '8', '9', 'C' },

{ '\*', '0', '#', 'D' }

};

const byte ROW\_PINS[4] = { 5, 4, 3, 2 };

const byte COL\_PINS[4] = { 6, 7, 8, 13 };

Keypad heroKeypad = Keypad(makeKeymap(BUTTONS), ROW\_PINS, COL\_PINS, 4, 4);

const byte BUZZER\_PIN = 12;

const byte RED\_PIN = 11;

const byte GREEN\_PIN = 10;

const byte BLUE\_PIN = 9;

void setup() {

pinMode(RED\_PIN, OUTPUT);

pinMode(GREEN\_PIN, OUTPUT);

pinMode(BLUE\_PIN, OUTPUT);

Serial.begin(9600);

// Load the PIN from EEPROM

if (loadPINFromEEPROM()) {

Serial.println("Stored PIN loaded successfully.");

isLocked = true;

displayColor(128, 0, 0); // Red LED indicates locked state

} else {

Serial.println("No PIN found in memory. Set a new PIN (4-8 characters). End with #:");

setupNewPIN(); // Prompt to set new PIN if no valid PIN is in EEPROM

}

}

void loop() {

if (isLocked) {

Serial.println("System is locked. Enter PIN to unlock:");

if (validatePIN()) {

isLocked = false;

Serial.println("System Unlocked! Options: '#' to lock, '\*' to change PIN, any other key to work.");

displayColor(0, 128, 0); // Green LED indicates unlocked state

} else {

attemptCount++;

if (attemptCount >= maxAttempts) {

Serial.println("Too many failed attempts. Access locked permanently.");

while (true); // Lock indefinitely

}

}

} else {

char button\_character = heroKeypad.getKey();

if (button\_character) {

if (button\_character == '#') { // Lock the system

isLocked = true;

Serial.println("System locked. Enter PIN to unlock.");

displayColor(128, 0, 0); // Red LED for locked state

} else if (button\_character == '\*') { // Change PIN

Serial.println("Enter current PIN to change it:");

if (validatePIN()) {

Serial.println("PIN accepted. Setting up new PIN.");

displayColor(128, 80, 0); // Yellow LED while changing PIN

setupNewPIN();

Serial.println("New PIN set successfully. System unlocked.");

displayColor(0, 128, 0); // Green LED for unlocked state

} else {

Serial.println("Incorrect PIN. Returning to unlocked mode.");

displayColor(0, 128, 0); // Keep green LED for unlocked state

}

} else { // Any other character to "work"

Serial.print("Working... You pressed: ");

Serial.println(button\_character);

giveInputFeedback(); // Short beep and blue LED for each work action

displayColor(0, 128, 0); // Maintain green LED for unlocked state

}

}

}

}

// Set up a new PIN with verification

void setupNewPIN() {

char tempPIN[maxPinLength + 1];

Serial.println("Enter new PIN:");

if (getPINInput(tempPIN)) {

Serial.println("Re-enter new PIN for verification:");

char verifyPIN[maxPinLength + 1];

if (getPINInput(verifyPIN) && strcmp(tempPIN, verifyPIN) == 0) {

strcpy(currentPIN, tempPIN);

savePINToEEPROM(); // Save the new PIN to EEPROM

Serial.println("New PIN set successfully!");

giveSuccessFeedback();

} else {

Serial.println("PINs did not match. Restarting setup.");

setupNewPIN();

}

}

}

// Save the current PIN to EEPROM

void savePINToEEPROM() {

int length = strlen(currentPIN);

EEPROM.write(eepromStartAddress, length); // Store the length of the PIN

for (int i = 0; i < length; i++) {

EEPROM.write(eepromStartAddress + 1 + i, currentPIN[i]);

}

}

// Load the PIN from EEPROM

bool loadPINFromEEPROM() {

int length = EEPROM.read(eepromStartAddress); // Read the PIN length

if (length < minPinLength || length > maxPinLength) {

return false; // No valid PIN stored

}

for (int i = 0; i < length; i++) {

currentPIN[i] = EEPROM.read(eepromStartAddress + 1 + i);

}

currentPIN[length] = '\0'; // Null-terminate the PIN

return true;

}

// Get PIN input with "#" to end entry

bool getPINInput(char\* pinBuffer) {

int charIndex = 0;

while (true) {

char key = heroKeypad.getKey();

if (key) {

if (key == '#') {

pinBuffer[charIndex] = '\0';

if (charIndex >= minPinLength && charIndex <= maxPinLength) {

return true;

} else {

Serial.println("Invalid PIN length. Try again.");

return false;

}

}

if (charIndex < maxPinLength) {

pinBuffer[charIndex] = key;

Serial.print("\*");

charIndex++;

}

giveInputFeedback();

}

}

}

bool validatePIN() {

char enteredPIN[maxPinLength + 1];

if (getPINInput(enteredPIN)) {

if (strcmp(enteredPIN, currentPIN) == 0) {

giveSuccessFeedback();

return true;

}

}

giveErrorFeedback();

return false;

}

void displayColor(byte red\_intensity, byte green\_intensity, byte blue\_intensity) {

analogWrite(RED\_PIN, red\_intensity);

analogWrite(GREEN\_PIN, green\_intensity);

analogWrite(BLUE\_PIN, blue\_intensity);

}

void giveInputFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 880, 200);

delay(200);

displayColor(0, 0, 128);

}

void giveSuccessFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 300, 200);

delay(200);

tone(BUZZER\_PIN, 500, 500);

delay(500);

displayColor(0, 128, 0);

}

void giveErrorFeedback() {

displayColor(0, 0, 0);

tone(BUZZER\_PIN, 300, 200);

delay(200);

tone(BUZZER\_PIN, 200, 500);

delay(500);

displayColor(128, 0, 0);

}

```

### Explanation of Changes

1. \*\*Save PIN to EEPROM\*\*:

- The `savePINToEEPROM()` function writes the PIN length and characters to EEPROM, allowing the system to remember the PIN across power cycles.

2. \*\*Load PIN from EEPROM\*\*:

- The `loadPINFromEEPROM()` function reads the stored PIN length and characters from EEPROM on startup. If a valid PIN is found, the system loads it and prompts for unlocking instead of setting a new PIN.

3. \*\*Startup Behavior\*\*:

- If a valid PIN is detected in EEPROM, the system skips the initial setup and prompts the user to unlock with the stored PIN. If no valid PIN is found, the user is prompted to set up a new one.

This should give you the persistence you want for the PIN across reboots. Let me know if there’s anything more to fine-tune!

Is the EEPROM Library included with the IDE or do I need to fetch and install it?

Yes, the \*\*EEPROM library\*\* is included with the Arduino IDE, so there’s no need to download or install it separately. Just add `#include <EEPROM.h>` at the top of your code, and you’re good to go!

The EEPROM library provides functions for reading, writing, and updating values in EEPROM, making it easy to store and retrieve data across power cycles. If you’re using Arduino IDE 1.6.5 or later, it’s built-in by default.

Let me know if you run into any questions while working with EEPROM!