MINISTERUL EDUCAŢIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică
Programul de studii: Tehnologia informației

RAPORT

Disciplina „IoT – Internetul Lucrurilor”
Tema: Interacțiunea cu utilizatorul

Student(ă): _____ **Vlașițchi Ștefan , TI-212**

Coordonator universitate: _____ **Lupan Cristian, asist.univ.**

Chișinău, 2024

1. **Serial STDIO**

Obiective:

1. Crearea unei aplicații MCU care să comunice cu utilizatorul prin intermediul unei interfețe seriale utilizând biblioteca STDIO.
2. Implementarea comenzilor pentru controlul unui LED, incluzând aprinderea și stingerea acestuia prin comenzi primite de la terminal.
3. Validarea funcționării aplicației prin simulare utilizând un mediu adecvat, precum Proteus, și separarea funcționalităților în fișiere pentru reutilizare ulterioară.

Introducere

Internetul Lucrurilor (IoT) reprezintă interconectarea dispozitivelor fizice prin intermediul internetului pentru a permite schimbul de date și interacțiunea automatizată între ele. În contextul acestui laborator, utilizarea microcontrolerelor (MCU) pentru a implementa interacțiunea cu un utilizator printr-o interfață serială este esențială pentru prototiparea și testarea aplicațiilor IoT. O problemă actuală abordată este controlul simplu al echipamentelor periferice, precum LED-uri, prin comenzi trimise prin terminal, permițând dezvoltatorilor să testeze rapid funcționalitatea fără a depinde de o interfață grafică. Referințe studiate includ documentația STDIO pentru microcontrolere și tutorialele de configurare IDE pentru dezvoltare și simulare.

Materiale si Metode utilizate

Materiale utilizate:

- Microcontroler cu suport pentru interfața serială(Arduino).
- LED și rezistor.
- IDE cu suport pentru Arduino și multiple fișiere, precum ArduinoIDE.
- Simulator Wokwi pentru validarea proiectului.

Metodologie:

- Configurarea MCU pentru a primi comenzi seriale utilizând biblioteca STDIO.
- Dezvoltarea unei logici pentru aprinderea și stingerea LED-ului pe baza comenzilor "led on" și "led off".
- Implementarea mesajelor de confirmare a comenzilor în terminal.
- Separarea funcționalităților într-un fișier dedicat pentru LED, astfel încât să fie reutilizabil în alte proiecte.

Rezultate

Schema bloc structurală:

Un bloc principal (Arduino UNO) pentru interacțiunea serială care primește comenzi de la terminal și decide asupra stării LED-ului (ON/OFF). Un al doilea bloc pentru controlul hardware al LED-ului.

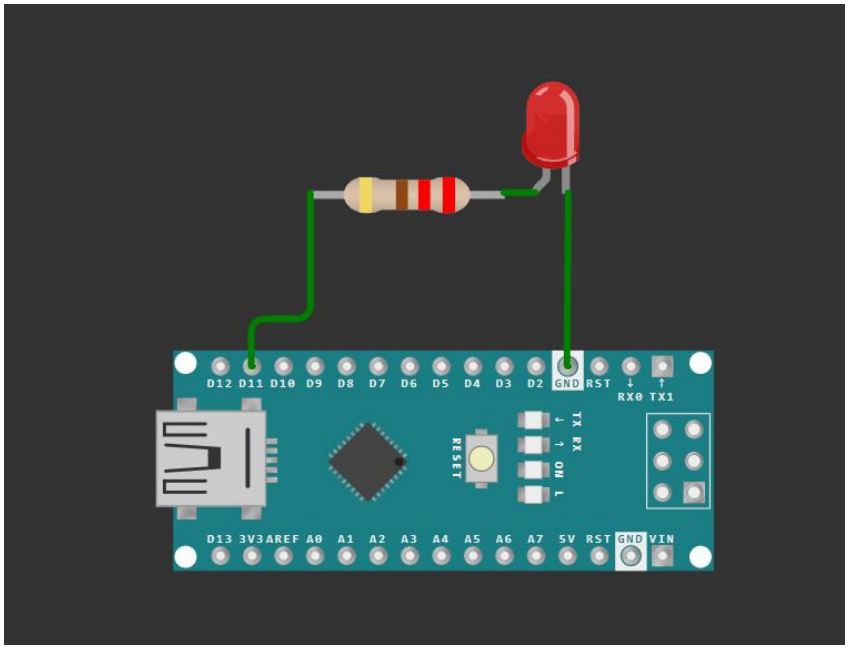


Figura 1.1 – Schema bloc structurala

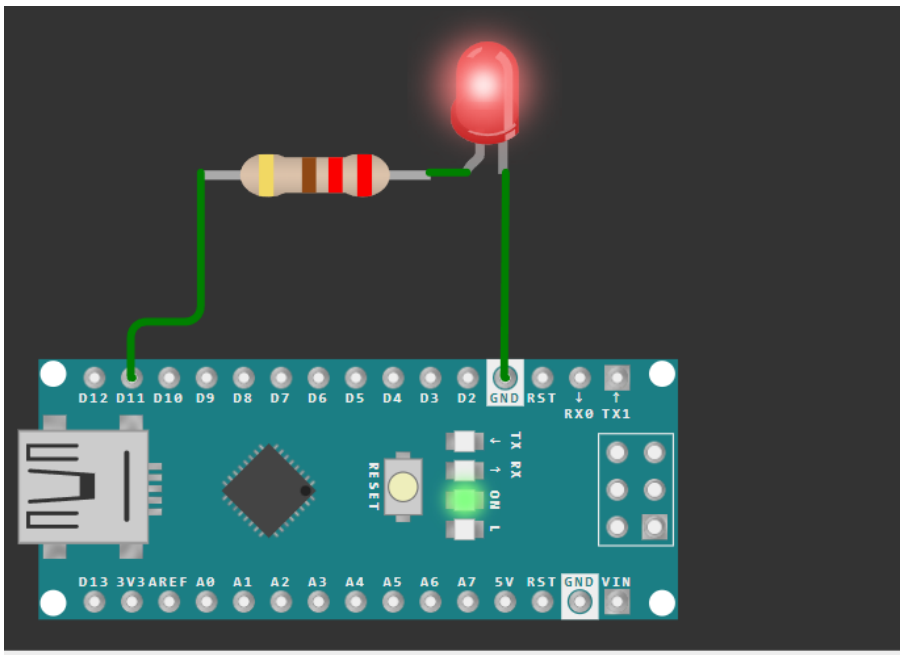
Cod Cheie:

```
// Inițializare LED
void led_init() {
   pinMode(LED_PIN, OUTPUT);
   digitalWrite(LED_PIN, LOW);
}

// Funcție pentru a seta starea LED-ului
void set_led_state(const char* command) {
   if (strcmp(command, "led on") == 0) {
      digitalWrite(LED_PIN, HIGH);
      printf("LED is ON\n");
   } else if (strcmp(command, "led off") == 0) {
      digitalWrite(LED_PIN, LOW);
      printf("LED is OFF\n");
   } else {
      printf("Invalid command\n");
   }
}

// În bucla principală
void loop() {
   char buffer[20];
   scanf("%s", buffer);
   set_led_state(buffer);
}
```

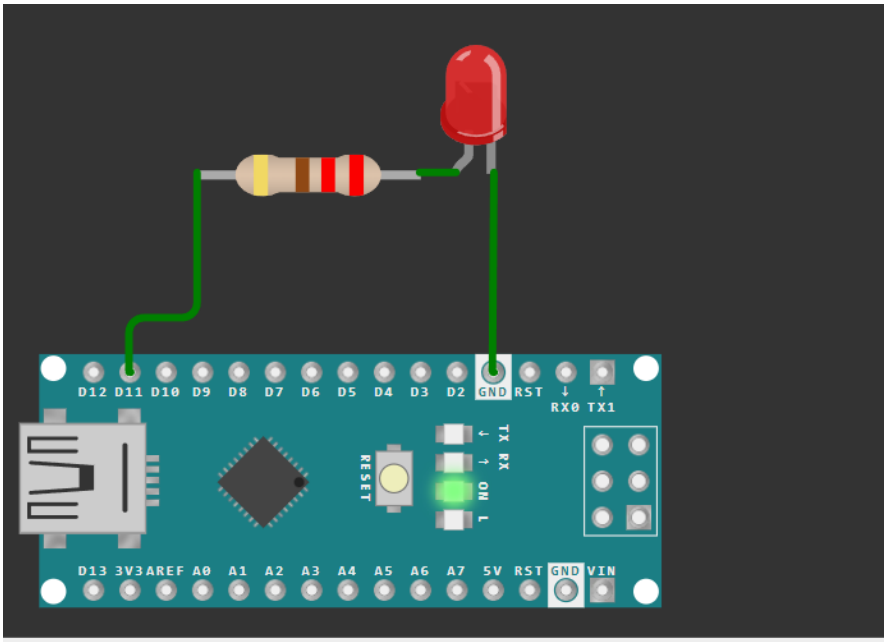
Această secvență de cod demonstreaza funcționalitatea esențială pentru aprinderea și stingerea LED-ului prin comenzi seriale și confirmarea comenzilor către utilizator.




```
System ready. Type 'led on' to turn the LED on or 'led off' to turn it off.
LED is now ON.
```

Figura 1.2 – Comanda pentru led-ul pornit

Pentru a aprinde ledul este necesar sa intrudcem comanda “**led on”** in consola, respective “**led off**” pentru a stinge led-ul




```
System ready. Type 'led on' to turn the LED on or 'led off' to turn it off.
LED is now ON.
LED is now OFF.
```

Figura 1.3 – Comanda pentru led-ul oprit

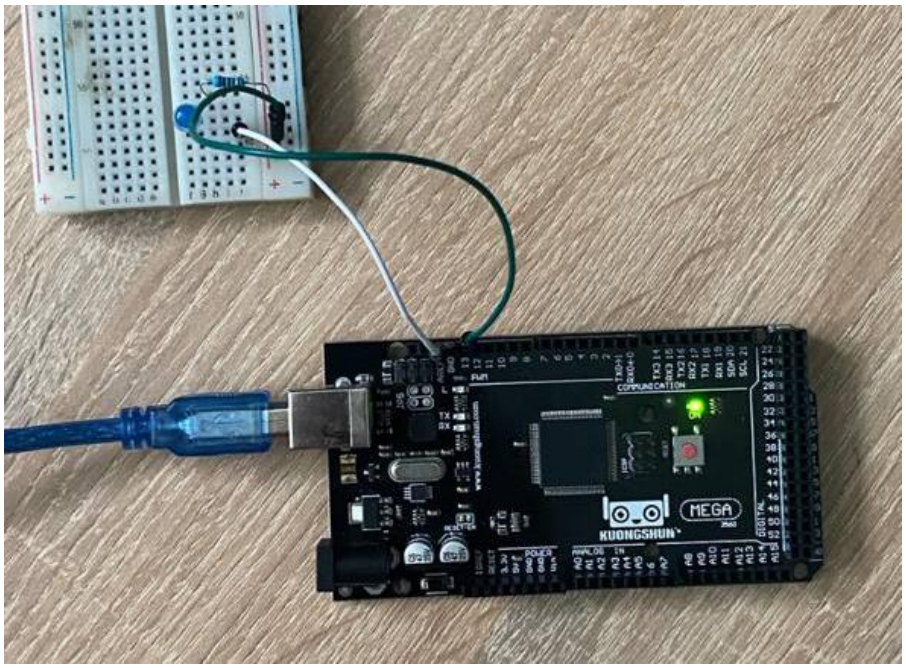


Figura 1.4 Schema fizica

Concluzie

Rezultatele obținute au demonstrat o interacțiune corectă între utilizator și sistem prin intermediul interfeței seriale. Comparativ cu alte soluții de control, acest proiect utilizează o metodă simplă și directă, fără utilizarea unor biblioteci complexe externe. Prin implementarea funcționalităților periferice în fișiere separate, codul este mai modular și ușor de reutilizat în proiecte viitoare. Utilizarea simulatorului Wokwi a oferit o modalitate eficientă de testare fără a avea nevoie de hardware fizic.

2. LCD + Keypad STDIO

Obiective

1. Configurarea unei aplicații MCU pentru a interacționa cu utilizatorul prin intermediul unei tastaturi 4x4 și a unui afișaj LCD, utilizând biblioteca STDIO.
2. Implementarea unui sistem de verificare a codurilor introduse de la tastatură, cu afișare a rezultatului pe un LCD.
3. Controlul unor LED-uri pentru a semnaliza codurile valide (LED verde) și codurile invalide (LED roșu).
4.

Introducere

Tastaturile și afișajele LCD sunt componente periferice esențiale în proiectele IoT, oferind o modalitate eficientă de interacțiune cu utilizatorii. În acest laborator, obiectivul este să folosim o tastatură 4x4 pentru a detecta coduri de acces și un LCD pentru a afișa feedback-ul corespunzător. Această soluție este utilă pentru

sistemele de acces și control, unde validarea unui cod este esențială. Sursele studiate includ documentația tastaturii matriciale și a afișajelor LCD, precum și utilizarea STDIO pentru manipularea intrărilor și ieșirilor textuale.

Materiale și Metode

Materiale utilizate:

- Microcontroler cu suport pentru interacțiunea serială și periferice.
- Tastatură 4x4 pentru introducerea codului.
- LCD 16x2 pentru afișarea mesajelor.
- LED verde și LED roșu pentru semnalizarea codului valid/invalid.
- IDE cu suport Arduino și STDIO
- Simulator Wokwi pentru testare.

Metodologie:

- Configurarea unui sistem de scanare a tastaturii 4x4 pentru a prelua codurile introduse.
- Implementarea unei logici de verificare a codului și afișarea rezultatelor pe LCD.
- Utilizarea LED-urilor pentru a semnaliza codul valid sau invalid.
- Separarea funcționalităților (scanare tastatură, afișare LCD, control LED-uri) în fișiere separate pentru reutilizare.

Rezultate

Schema bloc structurală:

Un bloc pentru scanarea tastaturii, unul pentru verificarea codului și un alt bloc pentru afișarea pe LCD. Blocurile LED sunt responsabile pentru semnalizarea stării.

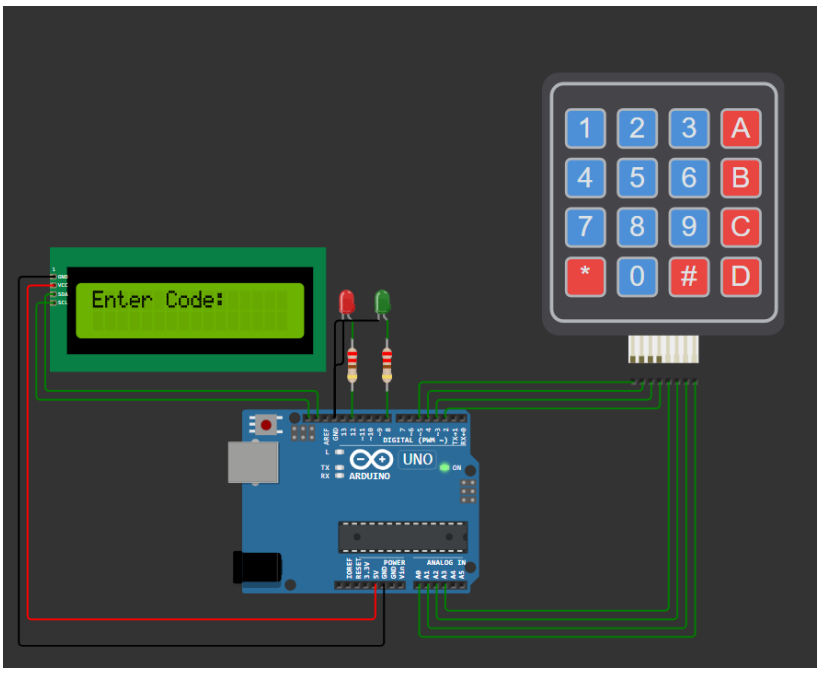


Figura 2.1 – Schema bloc structurala

Cod cheie

```
void loop() {
  char key = keypad.getKey();

  if (key) {
    if (key == '#') {
      // End of code entry
      lcd.clear();
      lcd.print("Code: ");
      lcd.print(inputCode);

      // Check if code is valid
      if (inputCode == validCode) {
        lcd.clear();
        lcd.print("Success");
        digitalWrite(greenLED, HIGH); // Turn on green LED
        digitalWrite(redLED, LOW);    // Turn off red LED
      } else {
        lcd.clear();
        lcd.print("Try Again");
        digitalWrite(greenLED, LOW);  // Turn off green LED
        digitalWrite(redLED, HIGH);   // Turn on red LED
      }

      delay(2000); // Display message for 2 seconds
      lcd.clear();
      lcd.print("Enter Code:");
      inputCode = ""; // Reset code
```

```
    } else if (key == '*') {
      // Clear input
      inputCode = "";
      lcd.clear();
      lcd.print("Enter Code:");
    } else {
      // Append the key to inputCode
      inputCode += key;
      lcd.setCursor(0, 1);
      lcd.print(inputCode);
    }
  }
}
```

Acest cod reflectă interacțiunea de bază dintre tastatură și afișajul LCD, precum și aprinderea LED-urilor corespunzătoare în funcție de validitatea codului.

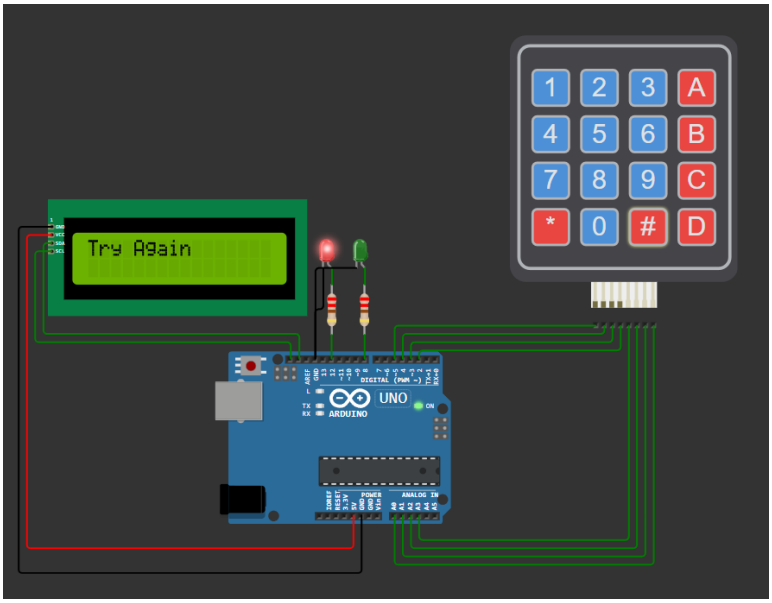


Figura 2.2 – Pin-ul introdus gresit

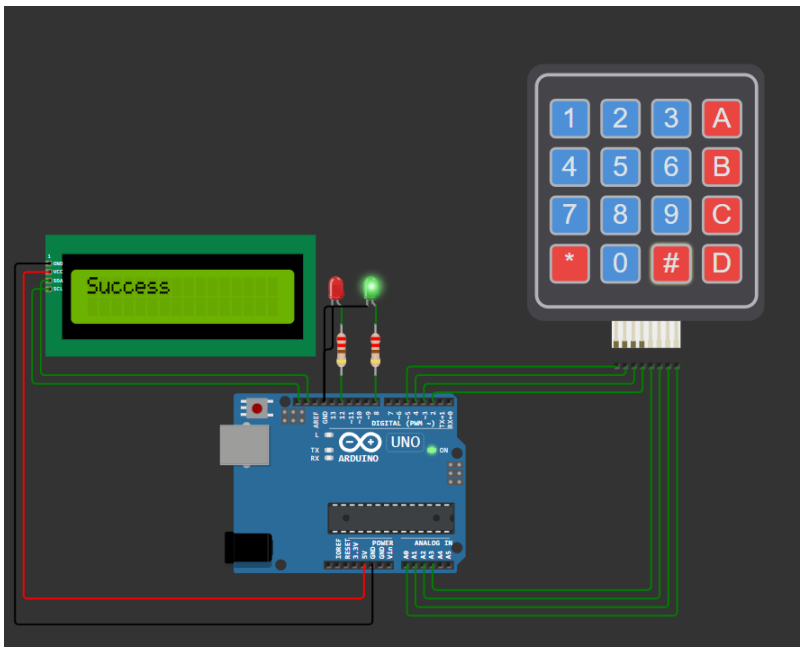


Figura 2.3 – Schema bloc structurala

Această implementare oferă o soluție eficientă pentru introducerea codurilor prin intermediul unei tastaturi și verificarea acestora. Comparativ cu alte metode de interacțiune, utilizarea tastaturii și LCD-ului oferă o metodă accesibilă pentru utilizator. Separarea funcționalităților (tastatură, afișare LCD și control LED-uri) în module separate contribuie la modularitatea și reutilizarea codului. Utilizarea simulatorului Proteus pentru testare a permis identificarea și remedierea erorilor înainte de implementarea pe hardware.

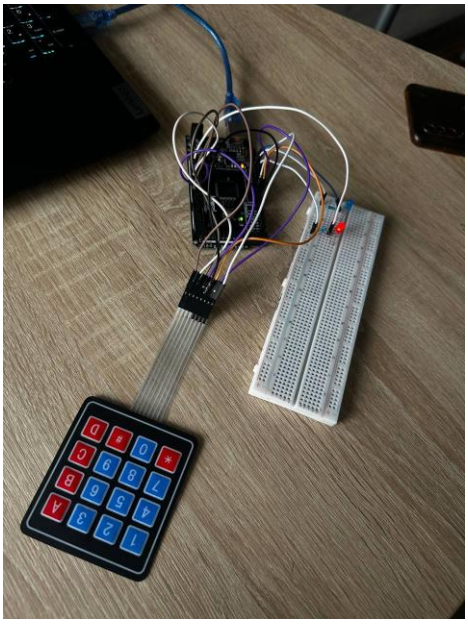


Figura 2.4 Schema fizica

Concluzie

Laboratorul a demonstrat implementarea unor sisteme de interacțiune cu utilizatorul prin intermediul serialului și perifericelor, folosind atât tastatura și LCD-ul, cât și comenzi textuale simple pentru controlul echipamentelor. Utilizarea bibliotecii STDIO a facilitat gestionarea eficientă a intrărilor și ieșirilor, iar modularitatea codului a permis separarea funcționalităților, contribuind astfel la reutilizarea lor în proiecte viitoare IoT. Testarea în simulare a validat corectitudinea soluțiilor propuse, oferind o bază solidă pentru extinderea acestora în sisteme mai complexe și eficiente.

Anexa A

stdinout.h

```
#ifndef _STDINOUT_H
#define _STDINOUT_H

// no need to make an instance of this yourself
class initializeSTDINOUT
{
        static size_t initnum;
public:
        // Constructor
        initializeSTDINOUT();
};

// Call the constructor in each compiled file this header is included in
// static means the names won't collide
static initializeSTDINOUT initializeSTDINOUT_obj;

#endif
```

stdinout.cpp

```
#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include <stdio.h>
#include "stdinout.h"

// Function that printf and related will use to print
static int serial_putchar(char c, FILE *f)
```

```
{
  if (c == '\n') {
    serial_putchar('\r', f);
  }

  return Serial.write(c) == 1 ? 0 : 1;
}
// Function that scanf and related will use to read
static int serial_getchar(FILE *)
{
  // Wait until character is avilable
  while (Serial.available() <= 0) { ; }

  return Serial.read();
}

static FILE serial_stdinout;

static void setup_stdin_stdout()
{
  // Set up stdout and stdin
  fdev_setup_stream(&serial_stdinout, serial_putchar, serial_getchar, _FDEV_SETUP_RW);
  stdout = &serial_stdinout;
  stdin  = &serial_stdinout;
  stderr = &serial_stdinout;
}

// Initialize the static variable to 0
size_t initializeSTDINOUT::initnum = 0;

// Constructor that calls the function to set up stdin and stdout
initializeSTDINOUT::initializeSTDINOUT()
{
  if (initnum++ == 0) {
    setup_stdin_stdout();
  }
}
```

Anexa B

main_1.ino

```
#include "stdinout.h"

const int LED_PIN = 11; // Pin for the LED

void setup() {
  Serial.begin(9600); // Start serial communication at 9600 baud
  pinMode(LED_PIN, OUTPUT); // Set LED pin as output

  printf("System ready. Type 'led on' to turn the LED on or 'led off' to turn it off.\n");
}

void loop() {
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n'); // Read the command until newline
    command.trim(); // Remove any leading or trailing whitespace

    if (command == "led on") {
      digitalWrite(LED_PIN, HIGH); // Turn LED on
      printf("LED is now ON.\n");
    } else if (command == "led off") {
      digitalWrite(LED_PIN, LOW); // Turn LED off
      printf("LED is now OFF.\n");
    } else {
      printf("Unknown command. Please type 'led on' or 'led off'.\n");
    }
  }
}
```

Anexa C

main_2.ino

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
#include "stdinout.h"

// Initialize the LCD with the I2C address 0x27 (common for many LCDs)
LiquidCrystal_I2C lcd(0x27, 16, 2); // 16 columns and 2 rows

// Define the keypad layout
const byte ROWS = 4; // four rows
const byte COLS = 4; // four columns

char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {5, 4, 3, 2}; // connect to the row pinouts of the keypad
byte colPins[COLS] = {A3, A2, A1, A0}; // connect to the column pinouts of the keypad

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

// Define LED pins
const int greenLED = 8;
const int redLED = 12;

String inputCode = ""; // To store the entered code
const String validCode = "123A"; // Define the valid code

void setup() {
  Serial.begin(9600);

  // Initialize LCD
  lcd.begin(16, 2); // 16 columns, 2 rows
  lcd.backlight(); // Turn on the backlight

  // Initialize LEDs
  pinMode(greenLED, OUTPUT);
  pinMode(redLED, OUTPUT);
  digitalWrite(greenLED, LOW);
  digitalWrite(redLED, LOW);

  lcd.print("Enter Code:");
}

void loop() {
  char key = keypad.getKey();

  if (key) {
    if (key == '#') {
      // End of code entry
      lcd.clear();
      lcd.print("Code: ");
      lcd.print(inputCode);

      // Check if code is valid
      if (inputCode == validCode) {
        lcd.clear();
        lcd.print("Success");
        digitalWrite(greenLED, HIGH); // Turn on green LED
        digitalWrite(redLED, LOW);    // Turn off red LED
```

```
      } else {
        lcd.clear();
        lcd.print("Try Again");
        digitalWrite(greenLED, LOW);  // Turn off green LED
        digitalWrite(redLED, HIGH);   // Turn on red LED
      }

      delay(2000); // Display message for 2 seconds
      lcd.clear();
      lcd.print("Enter Code:");
      inputCode = ""; // Reset code
    } else if (key == '*') {
      // Clear input
      inputCode = "";
      lcd.clear();
      lcd.print("Enter Code:");
    } else {
      // Append the key to inputCode
      inputCode += key;
      lcd.setCursor(0, 1);
      lcd.print(inputCode);
    }
  }
}
```