Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de laborator Nr.1
Disciplina: IoT
Tema: Interacțiunea cu Utilizatorul

A efectuat: st.gr.TI-212,
Muntean Mihai

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

Definirea problemei:

1. Să se proiecteze o aplicație în bază de MCU care ar schimba starea unui LED la detectarea apasării unui buton. De simulat funcționarea aplicației într-un simulator.
2. Să se proiecteze o aplicașție în bază de MCU care ar recepționa comenzi de la terminal prin interfața serială pentru a seta starea unui LED.
   - led on, pentru aprindere;
   - led off, pentru stingere;
   - sistemul trebuie să răspundă cu mesaje text de confirmare a comenzii;
   - pentru schimbul de text prin terminal a se utiliza librăria STDIO.
3. Să se proiecteze o aplicație în bază de MCU pentru detectarea unui cod de la o tastatura 4x4, să verifice codul și să afișeze mesaj la un LCD.
   - pentru cod valid sa se aprinda un led de culoare verde, pentru cod invalid, un led de culoare rosie;
   - a se utiliza **stdio.h** pentru scanarea tastaturii și afișare la LCD.

Obiective:

1. Inițierea în programarea MCU (Microcontroller Unit);
2. Proiectarea aplicațiilor în conformitate cu problemele relatate;
3. Ansamblarea unor circuite reale pe baza soluțiilor dezvoltate.

Introducere

Un Microcontroller Unit (MCU), sau microcontroler, este un sistem integrat care combină un procesor, memorie și periferice într-un singur cip. Aceste componente permit microcontrolerului să execute sarcini specifice și să controleze diverse dispozitive electronice. Datorită versatilității și costurilor reduse, microcontrolerele sunt utilizate pe scară largă în aplicații industriale, comerciale și de consum.

Aplicațiile bazate pe MCU sunt importante în dezvoltarea soluțiilor IoT (Internet of Things), deoarece permit interacțiunea dintre dispozitivele fizice și software-ul care le controlează. De exemplu, microcontrolerele pot fi utilizate pentru a monitoriza și controla echipamentele, a gestiona datele și a comunica cu alte dispozitive prin interfețe seriale sau digitale.

Această lucrare de laborator presupune proiectarea și dezvoltarea unor aplicații simple de interacțiune cu utilizatorul bazate pe MCU, având ca scop demonstrarea cum aceste dispozitive pot fi utilizate pentru a rezolva probleme specifice.

Informații despre metodologia de proiectare și ansamblare este studiată din resursa electronică WOKWI [1] și cursul de pe ELSE [2].

Rezultate obținute:

Sarcina 1

Această sarcină presupune crearea aplicații în bază de MCU, în care stările unui LED se modifică în funcție de apăsarea unui buton.

Ca materiale pentru simularea sistemului au fost utilizate:

- LED;
- Rezistor;
- Buton;
- Placa Arduino Nano;
- Fire de contact.

Ansamblarea sistemului începe cu conectarea led-ului la placă: catodul la GND, iar anodul prin conectarea în serie a rezistorului la unul din pinii digitali (D8). Butonul la rândul său are 2 seturi de pini 1(l & r) și 2(l & r), un set fiind conectat la GND, iar altul la unul din pinii digitali (D6). Curentul astfel circulă de la pinii digitali, care sunt configurați în cod `pinMode()`, spre GND.

Deoarece pinii digitali pot avea doar 2 stări, `HIGH`(5V sau 3.3V) și `LOW`(0V) se inițializează global starea butonului ca `LOW`, în urma apăsării acesta devine `HIGH`, stările se alternează, iar odată cu starea butonului se modifică și starea LED-ului care se reflectă cu aprinderea și stingerea acestuia.

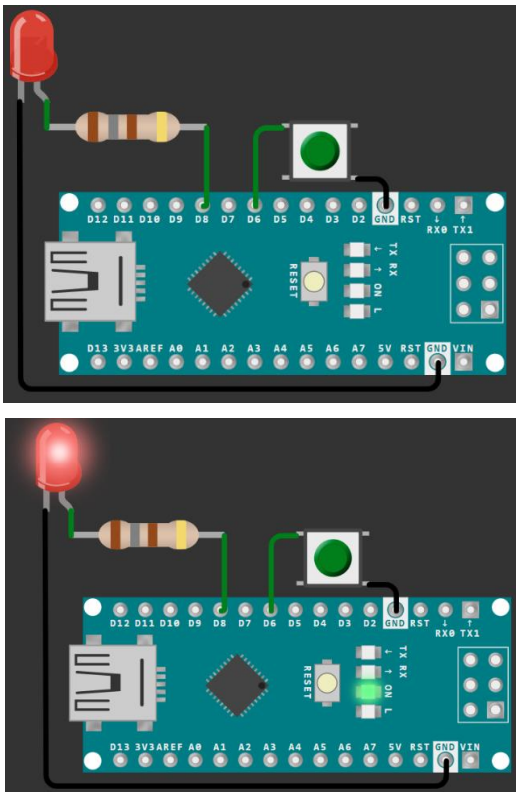


Figura 1 – Stările LED-ului, utilizare buton

Codul sursă:

```
#define LED_PIN 8
#define BUTTON_PIN 6

byte buttonState = LOW;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
}
void loop() {
  buttonState = digitalRead(BUTTON_PIN);
  buttonState == LOW ? digitalWrite(LED_PIN, LOW) :
              digitalWrite(LED_PIN, HIGH);
}
```

Sarcina 2

2.1 Configurarea aplicației pentru lucrul cu librăria stdio.h [3] vizualizare în *Anexa 1.1*

2.2 Această sarcină presupune crearea aplicații în bază de MCU, în care stările unui LED se modifică în funcție de comanda primită prin terminalul configurat pentru utilizarea librăriei STDIO.

Ca materiale pentru simularea sistemului au fost utilizate:

- LED;
- Rezistor;
- Placa Arduino Nano;
- Fire de contact.

Ansamblarea este una simplă, catodul led-ului se conecteaza la GND, iar anodul prin conexiune în serie cu rezistorul la unul din pinii digitali (D8).

După ce montajul hardware este complet, aplicația inițializează comunicarea serială cu o rată de transmisie de 115200 baud (bps). În funcția **setup()**, pinul digital asociat LED-ului (D8) este configurat ca OUTPUT, pentru a permite controlul stării acestuia.

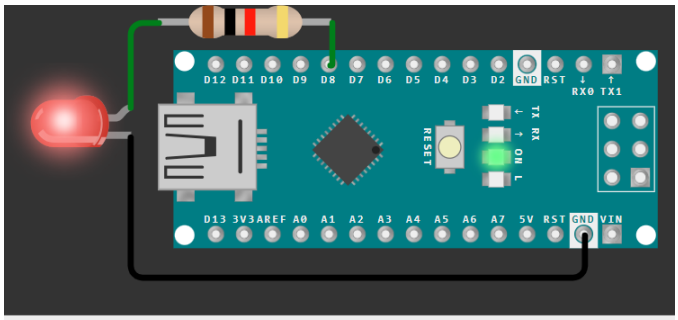
1. În bucla infinită **loop()**, aplicația apelează funcția **processCommand**(), care se ocupă de citirea comenzilor introduse prin terminalul serial, folosind funcția **fgets()**. Comanda este apoi procesată pentru a determina starea LED-ului: `fgets(command, sizeof(command), stdin);`
2. Comanda citită este prelucrată prin eliminarea caracterului de sfârșit de linie cu ajutorul funcției: `command[strcspn(command, "\n")] = 0;`
3. Comanda este apoi comparată cu cele așteptate, utilizând funcția **strcasecmp()** pentru a permite compararea insensibilă la majuscule:
4. În funcție de comanda primită, starea LED-ului este modificată folosind: `digitalWrite(LED_PIN, val ? HIGH : LOW);` Aceasta determină dacă LED-ul este aprins (`HIGH`) sau stins (`LOW`), iar informația este transmisă utilizatorului printr-un mesaj în terminal: `printf("LED is %s\n", val ? "ON" : "OFF");`

Codul sursă în *anexa 1.2*

```
void setup() {
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);

  printf("All staff is ready. Try \"led on\" or \"led off\" commands:\n");
}
```

```
void loop() {
  if (Serial.available()) {
    processCommand();
  }
}
```



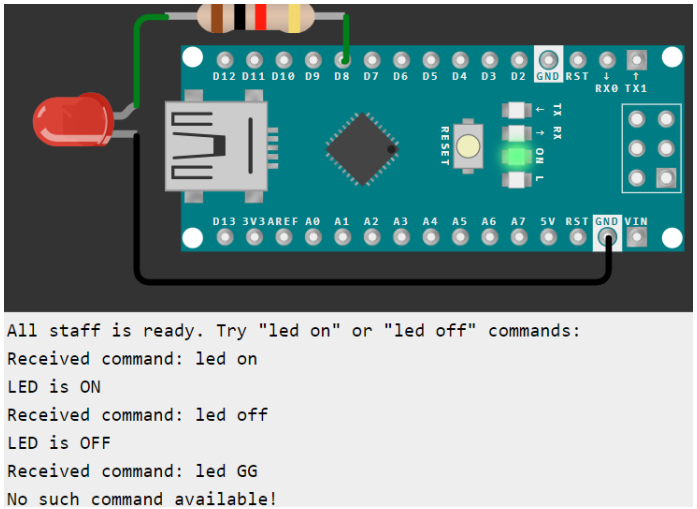


Figura 2 – Stările LED-ului, utilizare terminal

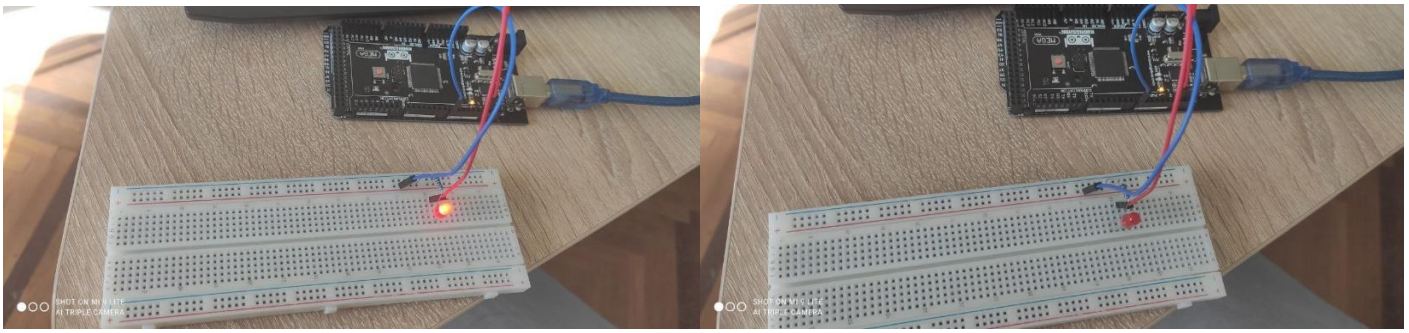


Figura 3 – Stările LED-ului, utilizare terminal, circuit fizic

Sarcina 3

Sarcina 3 presupune crearea unei aplicații în bază de MCU, care validează un cod inserat de la o tastatură 4x4, ca fiind adevărat sau greșit prin aprinderea unui LED specific completat de un mesaj pe LCD. Ca materiale pentru simularea sistemului au fost utilizate:

- 2 x LED;
- 2 x Rezistor;
- Placa Arduino Uno;
- LCD 16x2 (I2C);
- Keypad;

- Fire de contact.

Ansamblarea sistemului:

1. Ledurile sunt conectate cu catodul la GND iar anodul la unii din pinii digitali (D12 și D8);
2. Pentru LCD: SDA -> A4, SCL->A5, VCC->5V, GND->GND;
3. Keypad-ul 4x4:
    a. Pinurile rândurilor (R1 ,R2 ,R3 ,R4) la pinul plăcii respectiv (D5, D4, D3, D2);
    b. Pinurile coloanelor (C1, C2, C3, C4) la pinurile (A3, A2, A1, A0).

Algoritmul sistemului (Codul sursa în *anexa 1.3*) :

1. Inițializare:
    1.1. Se inițializează comunicația serială la 9600 bps ;
    1.2. Se configurează LCD-ul pentru afișaj și se activează iluminarea de fundal;
    1.3. Se configurează LED-urile pentru a funcționa ca ieșiri (pin D8 pentru LED verde și pin D12 pentru LED roșu);
    1.4. Se configurează tastatura 4x4 folosind pinii specificați pentru rânduri și coloane;
2. Configurare LCD (în initializeLcd):
    2.1. LCD-ul afișează mesajul „Guess the CODE!” și „4 digits” pentru a ghida utilizatorul;
    2.2. LCD-ul este curățat și pregătit pentru a accepta codul de la tastatură;
3. În bucla infinită (funcția **loop**()):
    3.1. Se așteaptă și se citește codul introdus de utilizator de la tastatura 4x4 (în **getEnteredCode**);
    3.2. Se determină dacă codul introdus este corect sau greșit (în **determineResult**);
    3.3. Se afișează rezultatul pe LCD și se aprinde LED-ul corespunzător pentru 2 secunde (în **displayResults**);
    3.4. LCD-ul este resetat pentru a permite introducerea unui nou cod;

```
void setup() {
  Serial.begin(9600);
  initializeLcd();
  pinMode(GREEN_LED_PIN, OUTPUT);
  pinMode(RED_LED_PIN, OUTPUT);
}
void loop() {
  String enteredCode = getEnteredCode(CODE_LENGTH);
  String message;
  uint8_t ledPin;
  determineResult(enteredCode, message, ledPin);
  displayResults(ledPin, message);
```

```
  resetLcd();
}
```

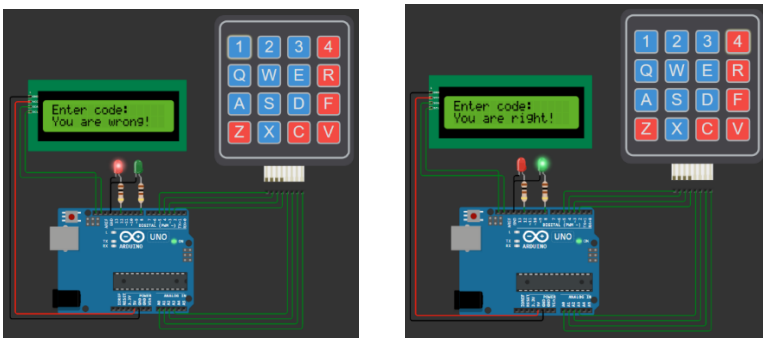


Figura 4 – Stările LED-ului cu afișare mesaj, utilizare tastatură 4x4

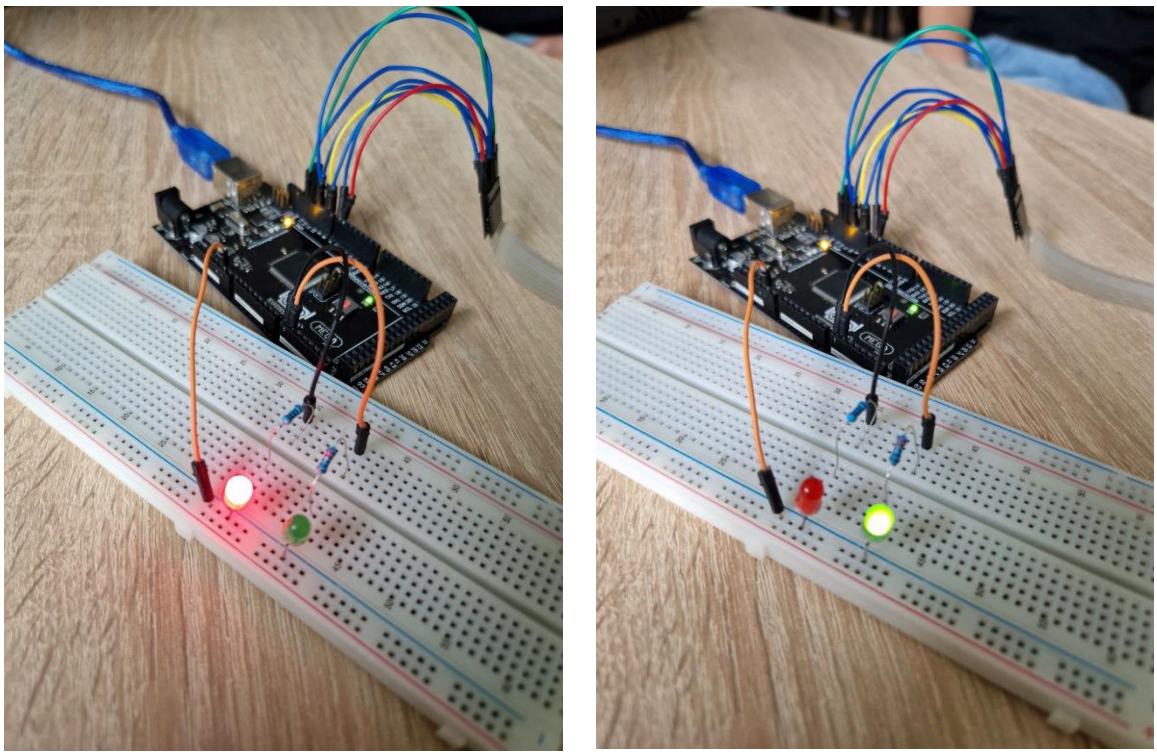


Figura 5 – Stările LED-ului, utilizare tastatură 4x4, circuit fizic

Concluzie

În cadrul acestei lucrări de laborator, am reușit să proiectăm și simulăm cu succes trei aplicații bazate pe microcontrolere (MCU) utilizând diverse componente electronice precum LED-uri, butoane, tastatură 4x4 și afișaj LCD (tip I2C). Am studiat metoda de ansamblare a acestor componente periferice cu placa Arduino astfel încât să putem crea un circuit fizic, iar aplicațiile create au demonstrat capacitatea microcontrolerelor de a interacționa cu dispozitive fizice, de a procesa comenzi și de a oferi feedback utilizatorului, contribuind la dezvoltarea soluțiilor IoT. Ce ține de simularea sistemului pe calculator, platforma Wokwi deține o bază solidă de cercetare, proiectare și simulare virtuală, cu o interfață prietenoasă și ușor de manipulat a sistemelor electronice.

Așa cum obiectivele stabilite inițial au fost acoperite, pot spune ca o concluzie, că ac activitatea realizată nu doar că a îmbunătățit abilitățile de proiectare și programare, dar ne-a și oferit o înțelegere practică a procesului de dezvoltare IoT, începând de la conectarea hardware și până la implementarea de aplicații software eficiente.

Bibliografii

1. Resursa electronică: https://docs.wokwi.com/?utm_source=wokwi – Regim de acces;
2. Resursa electronică: https://else.fcim.utm.md/course/view.php?id=343 – Regim de acces;
3. Resursa electronică: https://forum.arduino.cc/t/serial-print-and-printf/146256/14 - Regim de acces;

Anexa 1.1

Fișierul 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
```

Fișierul 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 1.2

Fișierul sketch.ino

```
#include "stdinout.h"
#define LED_PIN 8

void processCommand();
void handleLEDCommand();
void handleUnknownCommand();
void setLEDState(bool val);

char command[10];

void setup() {
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);
  printf("All staff is ready. Try \"led on\" or \"led off\" commands:\n");
}

void loop() {
  if (Serial.available()) {
    processCommand();
  }
}

void processCommand() {
  fgets(command, sizeof(command), stdin);
  command[strcspn(command, "\n")] = 0;     // Remove the newline character
  printf("Received command: %s\n", command);
  handleLEDCommand();
}

void handleLEDCommand() {
  if (strcasecmp(command, "led on") == 0) {
    setLEDState(true);
  }
  else if (strcasecmp(command, "led off") == 0) {
    setLEDState(false);
  }
  else {
    handleUnknownCommand();
  }
}

void handleUnknownCommand() {
  printf("No such command available!\n");
}

void setLEDState(bool val) {
  digitalWrite(LED_PIN, val ? HIGH : LOW);
  printf("LED is %s\n", val ? "ON" : "OFF");
}
```

Anexa 1.3

sketch.ino

```
#include "lcd_utils.h"
#include "keypad_utils.h"

#define GREEN_LED_PIN 8
#define RED_LED_PIN  12

LiquidCrystal_I2C lcd(I2C_ADDR, LCD_COLUMNS, LCD_LINES);
Keypad keypad = Keypad(makeKeymap(KEYPAD_KEYS), ROW_PINS, COL_PINS, ROWS, COLS);

const uint8_t CODE_LENGTH = 4;
const String CORRECT_CODE = "1234";

void displayResults(uint8_t ledPin, const String &message);
void determineResult(const String &enteredCode, String &message, uint8_t &ledPin);

void setup() {
  Serial.begin(9600);
  initializeLcd();
  pinMode(GREEN_LED_PIN, OUTPUT);
  pinMode(RED_LED_PIN, OUTPUT);
}

void loop() {
  String enteredCode = getEnteredCode(CODE_LENGTH);

  String message;
  uint8_t ledPin;

  determineResult(enteredCode, message, ledPin);

  displayResults(ledPin, message);
  resetLcd();
}

void displayResults(uint8_t ledPin, const String &message) {
  lcd.setCursor(0, 1);
  delay(500);
  lcd.print(message);
  digitalWrite(ledPin, HIGH);
  delay(2000);
  digitalWrite(ledPin, LOW);
}

void determineResult(const String &enteredCode, String &message, uint8_t &ledPin) {
  if (enteredCode == CORRECT_CODE) {
    message = "You are right!";
    ledPin = GREEN_LED_PIN;
  } else {
    message = "You are wrong!";
```

```
    ledPin = RED_LED_PIN;
  }
}
```

lcd_utilis.h

```
#ifndef LCD_UTILS_H
#define LCD_UTILS_H

#include <LiquidCrystal_I2C.h>
#include <Keypad.h>

#define I2C_ADDR    0x27
#define LCD_COLUMNS 16
#define LCD_LINES   2

extern LiquidCrystal_I2C lcd;

void initializeLcd();
void resetLcd();

#endif // LCD_UTILS_H
```

lcd_utilis.cpp

```
#include "lcd_utils.h"

void initializeLcd() {
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Guess the CODE!");

  String message = "4 digits";
  lcd.setCursor(4, 1);
  for(int i = 0; i < message.length(); i++){
    lcd.write(message[i]);
    delay(100);
  }

  delay(2000);
  resetLcd();
}

void resetLcd() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Enter code:");
  lcd.setCursor(5, 1);
  lcd.print("[____]");
}
```

keypad_utilis.h

```
#ifndef KEYPAD_UTILS_H
#define KEYPAD_UTILS_H

#include <Keypad.h>
#include "lcd_utils.h"

#define ROWS 4
#define COLS 4

const char KEYPAD_KEYS[ROWS][COLS] = {
  { '1', '2', '3', '4' },
  { 'Q', 'W', 'E', 'R' },
  { 'A', 'S', 'D', 'F' },
  { 'Z', 'X', 'C', 'V' }
};

const uint8_t COL_PINS[COLS] = { A3, A2, A1, A0 };
const uint8_t ROW_PINS[ROWS] = { 5, 4, 3, 2 };

extern Keypad keypad;

String getEnteredCode(uint8_t codeLength);

#endif // KEYPAD_UTILS_H
```

keypad_utilis.cpp

```
#include "keypad_utils.h"

String getEnteredCode(uint8_t code_length) {
  String code = "";
  while (code.length() < code_length) {
    char key = keypad.getKey();
    if (key != NO_KEY) {
      code += key;
      lcd.setCursor(code.length() + 5, 1);
      lcd.write(key);
    }
  }
  return code;
}
```