TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Report of laboratory work №1

Theme: User interaction

Fulfilled: st. gr. FAF-202 **Popa Eugeniu**

Controlled: univ. lecturer **Moraru Dumitru**

Chișinău 2023

THE TASKS OF THE LABORATORY WORK

Task 1: To design an MCU-based application that would change the status of an LED when a button press is detected.

Task 2: To design an MCU-based application that would receive commands from the terminal via the serial interface to set the status of an LED.

- *led on* for switching on and *led off* for switching off…, the system must respond with text messages about the order confirmation
- STDIO library for terminal text exchange has to be used

Task 3: To design an MCU-based application that would detect code from a 4x4 keyboard, verify it, and display a message on an LCD.

- for valid code to light a green LED, for invalid code, a red LED
- STDIO library to scan the keyboard and display information on the LCD has to be used

THE PROGRESS OF THE WORK

Task 1

1. Description of the main functions used to perform the task

To perform task 1, I have created 2 classes: “Button” and “Led”.

“Button.h” is used to work with the button component. It has a constructor, and several functions (Fig 1.1).

```
#ifndef BUTTON_H
#define BUTTON_H

#include <Arduino.h>

class Button
{
    private:
        int buttonPin;
    public:
        Button();
        void setPin(int pin);
        void setup();
        bool readButton();
};

#endif
```

Fig 1.1 Button.h – The button custom library

They are implemented in “Button.cpp” (Fig 1.2). “void setPin (int pin)” sets the pin of the button, “void setup()” sets the pin mode, and “bool readButton()” returns a boolean value. Is the button is pressed, it returns true. Else it returns false.

```
#include "Button.h"

Button::Button() {}

void Button::setPin(int pin)
{
  buttonPin = pin;
}

void Button::setup()
{
  pinMode(buttonPin, INPUT);
}

bool Button::readButton()
{
    if (digitalRead(buttonPin) == HIGH)
    {
      return true;
    } else
    {
      return false;
    }
}
```

Fig 1.2 Button.cpp – Implementation of the button custom library

“Led.h” is used to work with the led component. It has a constructor, and several functions (Fig 1.3).

```
#ifndef LED_H
#define LED_H

#include <Arduino.h>

class Led
{
    private:
        int ledPin;
    public:
        Led();
        void setPin(int pin);
        void setup();
        void switchLight(bool ledState);
};

#endif
```

Fig 1.3 Led.h – The led custom library

They are implemented in “Led.cpp” (Fig 1.4). “void setPin (int pin)” sets the pin of the led, “void setup()” sets the pin mode, and “bool switchLight(bool ledState)” turns the led on or off depending on the parameter.

```
#include "Led.h"

Led::Led() {}

void Led::setPin(int pin)
{
  ledPin = pin;
}

void Led::setup()
{
  pinMode(ledPin, OUTPUT);
}

void Led::switchLight(bool ledState)
{
    bool state = ledState;
    if (state == true)
    {
      digitalWrite(ledPin, HIGH);
    } else
    {
      digitalWrite(ledPin, LOW);
    }
}
```

Fig 1.4 Led.cpp - Implementation of the led custom library

The main file “main.cpp” (Fig 1.5) creates the objects, sets them up, and runs the loop to switch the led by clicking the button.

```ino
#include "Button.h"
#include "Led.h"

#include <Arduino.h>

const int buttonPin = 7;
const int ledPin = 13;

byte lastButtonState = false;
byte ledState = LOW;

Button button;
Led led;

void setup()
{
  button.setPin(buttonPin);
  led.setPin(ledPin);
  button.setup();
  led.setup();
}

void loop()
{
  bool buttonState = button.readButton();
  if (buttonState != lastButtonState)
  {
    lastButtonState = buttonState;
    if (buttonState == false)
    {
      if (ledState == true)
      {
        ledState = false;
      } else
      {
        ledState = true;
      }
      led.switchLight(ledState);
    }
  }
}
```

Fig 1.5 main.cpp – The main file of the program

2. Electrical scheme made in Proteus

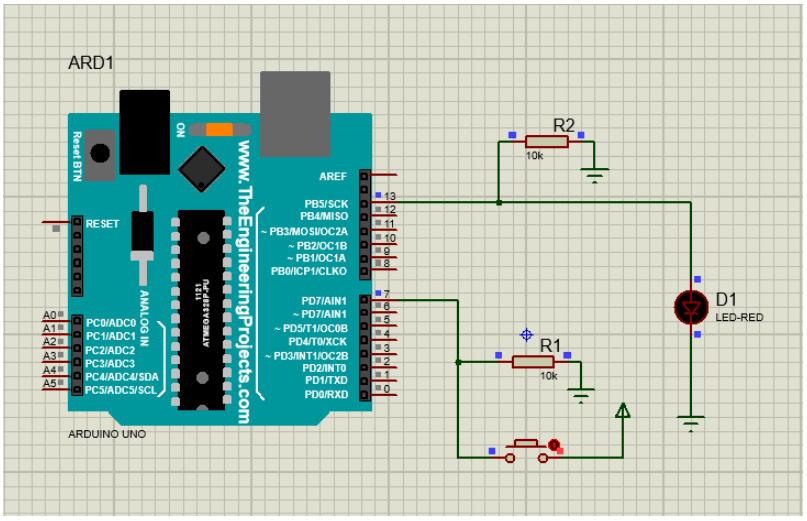

Fig 1.6 Electrical scheme for Task 1

3. Block diagram of the program

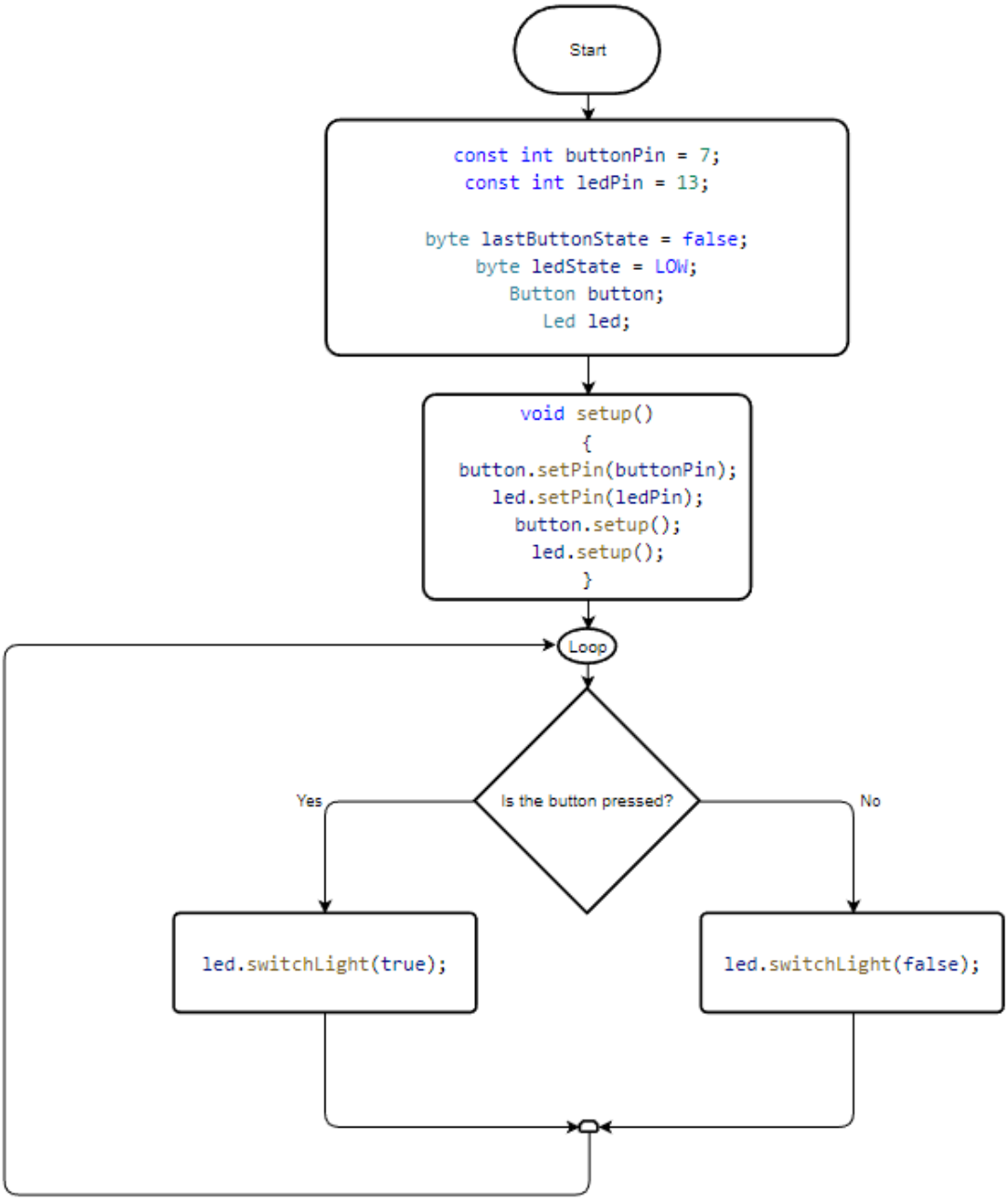

Fig 1.7 Flowchart of the program for Task 1

Task 2

1. Description of the main functions used to perform the task

To perform task 2, I have created 2 classes: “SInterface” and “Led”. “Led” is from the previous task. “SInterface.h” (Fig 1.1) is used to handle the serial input, to give commands to the led, and to output messages about the led state and message receival. It has a constructor and some functions.

```
#ifndef SINTERFACE_H
#define SINTERFACE_H

#include <Arduino.h>
#include <stdio.h>

class SerialInterface
{
private:
    byte ledState;
    String startOutput = "Hello! This is task 2";
    String askOutput = "Type 'turn on' to turn on the led, or 'turn off' to turn it off";
    String waitOutput = "Waiting for a command: ";
    int charsRead;
    char input[10];
    String turnOnMessage = "turn on";
    String receivedTurnOn = "Message received! Turning on the led";
    String alreadyOn = "The led is already on";
    String turnOffMessage = "turn off";
    String receivedTurnOff = "Message received! Turning off the led";
    String alreadyOff = "The led is already off";
    String wrongCommand = "There is no such command";

public:
    String receivedInput;
    SerialInterface();
    void setup();
    void waitInput();
    void takeInput();
    bool changeLight();
};

#endif
```

Fig 2.1 SInterface.h - The serial interface custom library

These functions are implemented in “SInterface.cpp” (Fig. 1.2). The constructor sets the led state to low by default. “setup()” sets the interface. “waitInput()” waits for the input from the interface and prints it if there is one. “takeInput()” takes the input of max.10 characters, than prints the “receivedInputMessage”. “bool changeLight()” returns true if the ledState is false and sets the state to true, and vice versa.

```cpp
#include "SInterface.h"
#include "Led.h"

SerialInterface::SerialInterface()
{
  ledState = LOW;
}

void SerialInterface::setup()
{
  Serial.begin(9600);
  Serial.println(startOutput);
  Serial.println(askOutput);
}

void SerialInterface::waitInput()
{
  Serial.println(waitOutput);
  while (Serial.available() == 0)
  {
  }
}

void SerialInterface::takeInput()
{
  delay(64);
  charsRead = Serial.readBytesUntil('\n', input, sizeof(input) - 1);
  input[charsRead] = 0;
  receivedInput = input;
  Serial.println(receivedInput);
}

bool SerialInterface::changeLight()
{
  if (receivedInput == turnOnMessage)
  {
    if (ledState == false)
    {
      ledState = true;
      Serial.println(receivedTurnOn);
      return true;
    }
    else
    {
      Serial.println(alreadyOn);
    }
  }
  else if (receivedInput == turnOffMessage)
  {
    if (ledState == true)
    {
      ledState = false;
      Serial.println(receivedTurnOff);
      return false;
    }
    else
    {
      Serial.println(alreadyOff);
    }
  }
  else
    Serial.println(wrongCommand);
}
```

Fig 2.2 SInterface.cpp- Implementation of the serial interface custom library

The main file “main.cpp” (Fig 2.3) creates the objects, sets them up, and runs the loop to switch the led by the return value of the “changeLight()” function.

```cpp
#include "SInterface.h"
#include "Led.h"

#include <Arduino.h>
#include <stdio.h>

const int ledPin = 13;

SerialInterface interface;
Led led;

void setup()
{
  led.setPin(ledPin);
  led.setup();
  interface.setup();
}

void loop()
{
  interface.waitInput();
  delay(64);
  interface.takeInput();
  bool result = interface.changeLight();
  if (result == true)
  {
    led.switchLight(true);
  }
  else if (result == false)
  {
    led.switchLight(false);
  }
}
```

Fig 2.3 main.cpp - The main file of the program

2. Electrical scheme made in Proteus

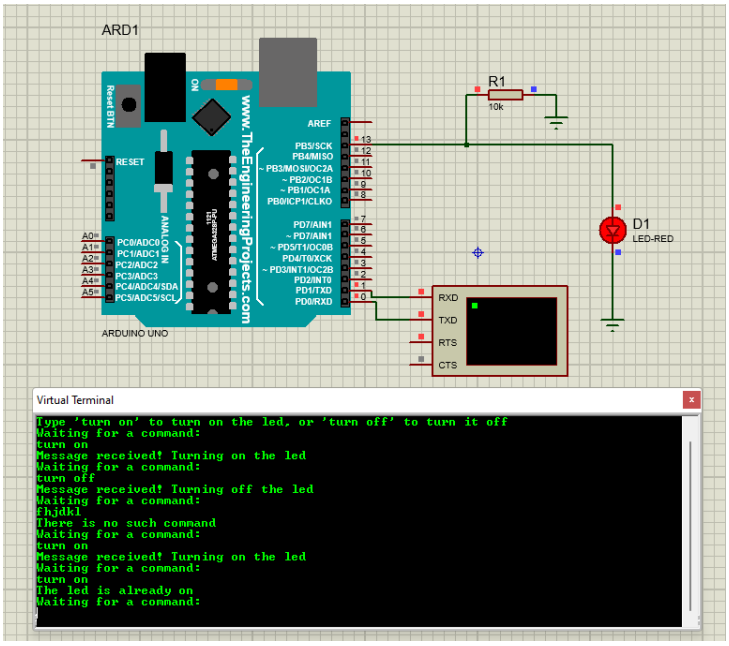

Fig 2.4 Electrical scheme for Task 2

3. Block diagram of the program

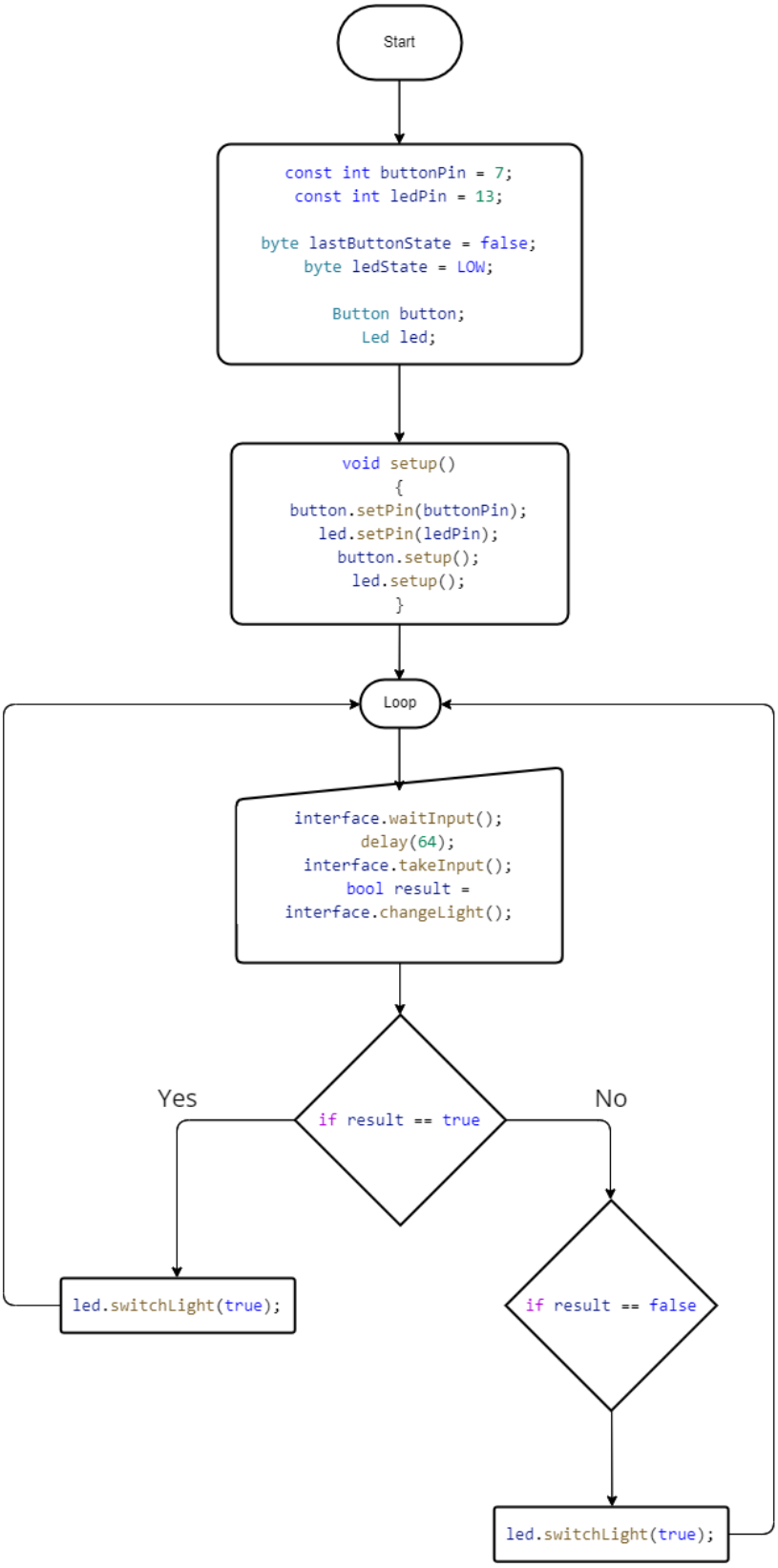

Fig 2.5 Flowchart for Task 2

Task 3

1. Description of the main functions used to perform the task

To perform this task, one third-party library for the keypad is used: “Keypad.h” [1]; Also, I have created my own library “CustomKeypad”.

“CustomKeypad.h” (Fig 3.1) is a library that handles the input from the keypad, checks the password matching, gives commands to the leds, outputs messages to the terminal. “SInterface” and “Led” are from the previous tasks.

```
#ifndef CUSTOMKEYPAD_H
#define CUSTOMKEYPAD_H

#include <Arduino.h>
#include <Keypad.h>

class CustomKeypad
{
    private:
        char greenLedCode[4] = {'2', '4', '6', '8'};
        const byte rows = 4;
        const byte cols = 4;
        char hexKeys[4][4] = {
            {'7', '8', '9', '/'},
            {'4', '5', '6', '*'},
            {'1', '2', '3', '-'},
            {'C', '0', '=', '+'}};
        byte rowPins[4] = {8, 9, 10, 11};
        byte colPins[4] = {2, 3, 4, 5};
    public:
        CustomKeypad();
        Keypad customKey();
        bool checkCode(char arr[]);
};

#endif
```

Fig 3.1 CustomKeypad.h - The keypad custom library

The functions are implemented in “CustomKeyboard.cpp” (Fig 3.2). “customKey()” initialized the keyboard from library “Keypad.h”. “checkCode()” is used to check the input array of numbers. It checks if the input numbers are equal to the secret code, and returns true if yes, otherwise false. In the main function this result will turn on the green or the red light.

```cpp
#include "CustomKeypad.h"

CustomKeypad::CustomKeypad() {}

Keypad CustomKeypad::customKey()
{
    Keypad customPad = Keypad(makeKeymap(hexKeys), rowPins, colPins, rows, cols);
    return customPad;
}

bool CustomKeypad::checkCode(char arr[])
{
    bool isEqualToGreen = true;
    for (int i = 0; i < 4; i++)
    {
        if (arr[i] != greenLedCode[i])
        {
            isEqualToGreen = false;
        }
    }
    if (isEqualToGreen == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Fig 3.2 CustomKeypad.cpp - Implementation of the keypad custom library

“Led” library has one more function “switchLightTimer()” (Fig 3.3) which turns the led for a specified period of time.

```cpp
void Led::switchLightTimer()
{
  digitalWrite(ledPin, HIGH);
  delay(3000);
  digitalWrite(ledPin, LOW);
}

bool Led::getLedState()
{
  if (ledState == false)
  {
      return false;
  }
  else
  {
    return true;
  }
}

void Led::setLedState(bool state)
{
  ledState = state;
}
```

Fig 3.3 Function from Led.cpp - Implementation of the led custom library

The main file “main.cpp” (Fig 3.4) creates the objects, sets them up, and runs the loop where the keyboard waits for the input, takes an array of for numbers, then checks the code. If the code is correct, it lights the green led, otherwise the red. It also outputs messages to the terminal if the code is correct or not.

```
#include "CustomKeypad.h"
#include "SInterface.h"
#include "Led.h"

#include <Arduino.h>
#include <stdio.h>

const int redLedPin = 13;
const int greenLedPin = 12;

int lastNumber = 0;
char codeArray[4];

CustomKeypad keypad;
Keypad customKeypad = keypad.customKey();
SerialInterface interface;
Led redLed;
Led greenLed;

void setup()
{
  interface.setup();
  redLed.setPin(redLedPin);
  redLed.setup();
  greenLed.setPin(greenLedPin);
  greenLed.setup();
}

void loop()
{
  char customKey = customKeypad.getKey();
  if (customKey)
  {
    if (lastNumber < 4)
    {
      Serial.print(customKey);
      codeArray[lastNumber] = customKey;
      lastNumber++;
      if (lastNumber == 4)
      {
        Serial.println("\n");
        Serial.println("Checking...");
        bool checkResult = keypad.checkCode(codeArray);
        if (checkResult == true)
        {
          Serial.println("The code is correct!");
          greenLed.switchLightTimer();
        }
        else if (checkResult == false)
        {
          Serial.println("The code is incorrect!");
          redLed.switchLightTimer();
        }
        strcpy(codeArray, "");
        lastNumber = 0;
      }
    }
  }
}
```

Fig 3.4 main.cpp - The main file of the program

2. Electrical scheme

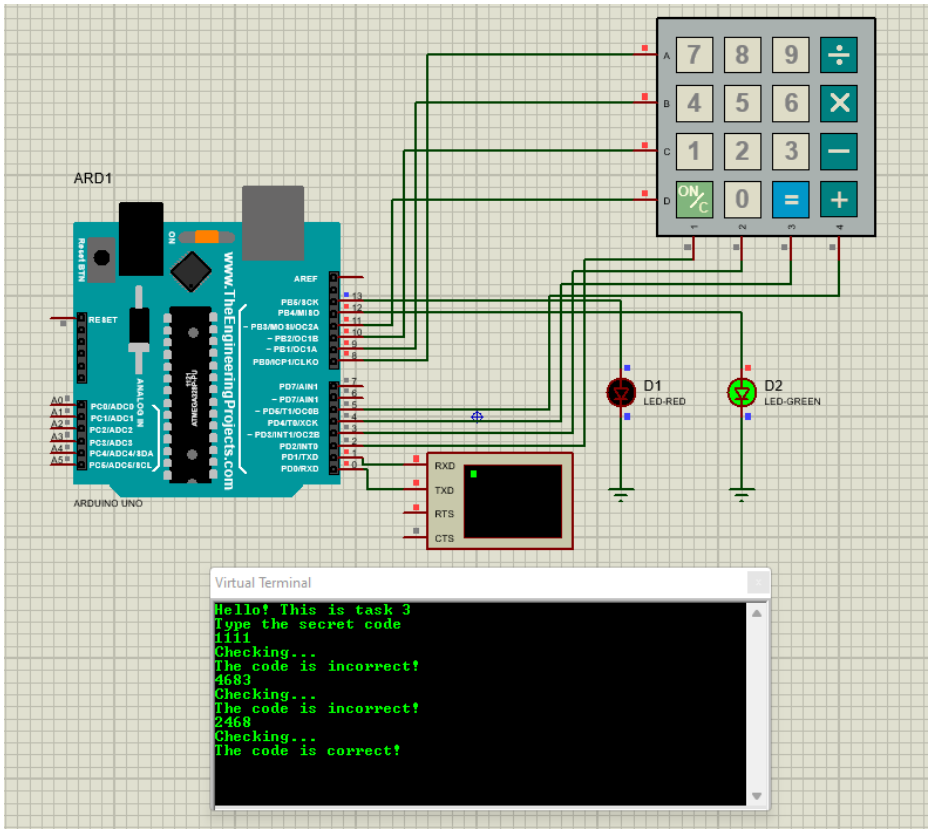

Fig 3.5 Electrical scheme for Task 3

3. Block diagram of the program

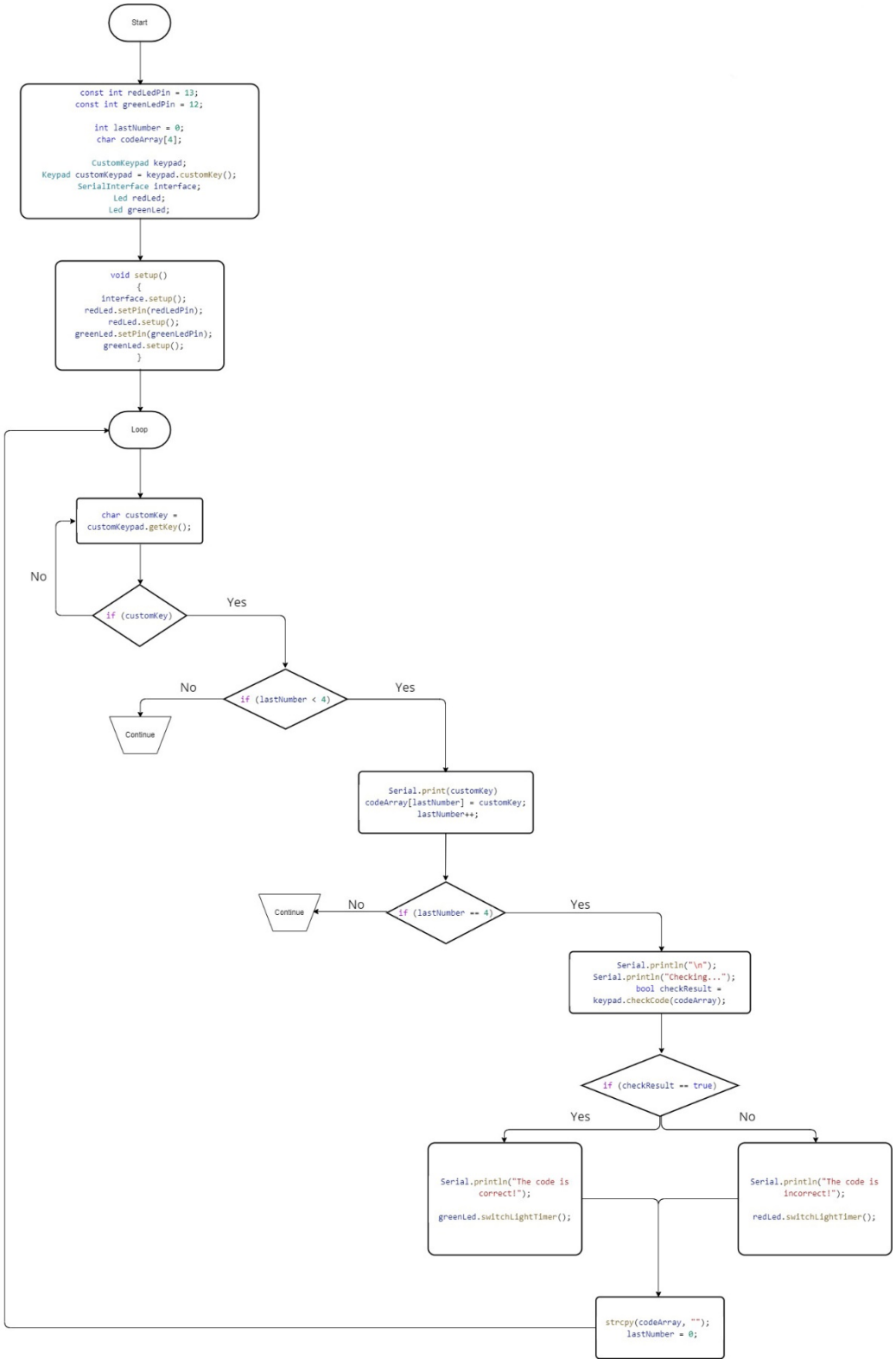

Fig 3.6 Flowchart for Task 3

CONCLUSIONS

In this laboratory work, I have learned the basics of programming on C++ for Arduino. Also, I have learned about simulating hardware in Proteus, which was used for testing and designing electrical schemes.

It has been a valuable learning experience. I have gained hands-on experience in designing and implementing microcontroller-based applications using various components such as LEDs, buttons, serial interface, keyboard, etc. The tasks have allowed me to apply the theoretical concepts that I have learned in the classroom to real-world applications and have helped me to develop my coding and problem-solving skills. I have learned how to use different libraries and programming techniques to control and monitor various physical devices and interfaces. The tasks have also taught me how to work with multiple components and communication interfaces, such as the serial interface, which has given me a deeper understanding of microcontroller-based systems. Overall, this laboratory work has been a challenging but rewarding experience. I feel more confident in my ability to design and implement microcontroller-based applications and I am grateful for the opportunity to apply my knowledge in a practical setting.

Bibliography

[1] Keypad: official documentation page of the library. Access link: Arduino Playground - Keypad Library

Tables

Table 1 Arduino Uno technical specifications

Numele parametrului	Valoarea
Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g