**Клейменов Тимофей,**

Новороссийский колледж строительства и экономики

Руководитель **Белова С.В., преподаватель**

**«АВТОМАТИЗАЦИЯ РАБОТЫ ТЕХНИЧЕСКОГО ОБЪЕКТА НА БАЗЕ ПРОГРАММИРУЕМОГО МИКРОКОНТРОЛЛЕРА»**

Это свободная программа: вы можете перераспространять ее и/или изменять ее на условиях Стандартной общественной лицензии GNU в том виде, в каком она была опубликована Фондом свободного программного обеспечения; либо версии 3 лицензии, либо (по вашему выбору) любой более поздней версии.

Эта программа распространяется в надежде, что она будет полезной, но БЕЗО ВСЯКИХ ГАРАНТИЙ; даже без неявной гарантии ТОВАРНОГО ВИДА или ПРИГОДНОСТИ ДЛЯ ОПРЕДЕЛЕННЫХ ЦЕЛЕЙ. Подробнее см. в Стандартной общественной лицензии GNU.

Вы должны были получить копию Стандартной общественной лицензии GNU вместе с этой программой. Если это не так, см. <https://www.gnu.org/licenses/>.

**GNU LESSER GENERAL PUBLIC LICENSE**

**Version 3, 29 June 2007**

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies

of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates

the terms and conditions of version 3 of the GNU General Public

License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser

General Public License, and the "GNU GPL" refers to version 3 of the GNU

General Public License.

"The Library" refers to a covered work governed by this License,

other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided

by the Library, but which is not otherwise based on the Library.

Defining a subclass of a class defined by the Library is deemed a mode

of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an

Application with the Library. The particular version of the Library

with which the Combined Work was made is also called the "Linked

Version".

The "Minimal Corresponding Source" for a Combined Work means the

Corresponding Source for the Combined Work, excluding any source code

for portions of the Combined Work that, considered in isolation, are

based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the

object code and/or source code for the Application, including any data

and utility programs needed for reproducing the Combined Work from the

Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License

without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a

facility refers to a function or data to be supplied by an Application

that uses the facility (other than as an argument passed when the

facility is invoked), then you may convey a copy of the modified

version:

a) under this License, provided that you make a good faith effort to

ensure that, in the event an Application does not supply the

function or data, the facility still operates, and performs

whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of

this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from

a header file that is part of the Library. You may convey such object

code under terms of your choice, provided that, if the incorporated

material is not limited to numerical parameters, data structure

layouts and accessors, or small macros, inline functions and templates

(ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the

Library is used in it and that the Library and its use are

covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license

document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that,

taken together, effectively do not restrict modification of the

portions of the Library contained in the Combined Work and reverse

engineering for debugging such modifications, if you also do each of

the following:

a) Give prominent notice with each copy of the Combined Work that

the Library is used in it and that the Library and its use are

covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license

document.

c) For a Combined Work that displays copyright notices during

execution, include the copyright notice for the Library among

these notices, as well as a reference directing the user to the

copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this

License, and the Corresponding Application Code in a form

suitable for, and under terms that permit, the user to

recombine or relink the Application with a modified version of

the Linked Version to produce a modified Combined Work, in the

manner specified by section 6 of the GNU GPL for conveying

Corresponding Source.

1) Use a suitable shared library mechanism for linking with the

Library. A suitable mechanism is one that (a) uses at run time

a copy of the Library already present on the user's computer

system, and (b) will operate properly with a modified version

of the Library that is interface-compatible with the Linked

Version.

e) Provide Installation Information, but only if you would otherwise

be required to provide such information under section 6 of the

GNU GPL, and only to the extent that such information is

necessary to install and execute a modified version of the

Combined Work produced by recombining or relinking the

Application with a modified version of the Linked Version. (If

you use option 4d0, the Installation Information must accompany

the Minimal Corresponding Source and Corresponding Application

Code. If you use option 4d1, you must provide the Installation

Information in the manner specified by section 6 of the GNU GPL

for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the

Library side by side in a single library together with other library

facilities that are not Applications and are not covered by this

License, and convey such a combined library under terms of your

choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based

on the Library, uncombined with any other library facilities,

conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it

is a work based on the Library, and explaining where to find the

accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions

of the GNU Lesser General Public License from time to time. Such new

versions will be similar in spirit to the present version, but may

differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the

Library as you received it specifies that a certain numbered version

of the GNU Lesser General Public License "or any later version"

applies to it, you have the option of following the terms and

conditions either of that published version or of any later version

published by the Free Software Foundation. If the Library as you

received it does not specify a version number of the GNU Lesser

General Public License, you may choose any version of the GNU Lesser

General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide

whether future versions of the GNU Lesser General Public License shall

apply, that proxy's public statement of acceptance of any version is

permanent authorization for you to choose that version for the

Library.

**Исходный код программы**

Файл button.h

#ifndef BUTTON\_H

#define BUTTON\_H

#include <stdint.h>

#include <Arduino.h>

#include "pin.h"

#define LONG\_PRESS\_TIME 800

class Button {

private:

pin m\_pin;

uint32\_t m\_timer;

uint8\_t m\_long\_press;

public:

Button(const uint8\_t pin\_number, bool pin\_state = false);

void checkState();

inline uint8\_t pin\_num() { return m\_pin.pin\_num; }

inline uint8\_t state() { return m\_pin.state; }

inline uint8\_t is\_long\_press() { return m\_long\_press; }

};

#endif // BUTTON\_H

Файл button.cpp

#include "button.h"

Button::Button(const uint8\_t pin\_number, bool pin\_state = false):

m\_pin{pin\_number, pin\_state},

m\_timer{0},

m\_long\_press{0}

{

};

void Button::checkState() {

if (digitalRead(m\_pin.pin\_num) == HIGH) {

if (m\_pin.state == LOW) {

m\_pin.state = HIGH;

m\_timer = millis();

#ifdef DEBUG

Serial.print("Clicked- Btn:");

Serial.println(m\_pin.pin\_num);

#endif

}

if ((millis() - m\_timer > LONG\_PRESS\_TIME) && !(m\_long\_press)) {

m\_long\_press = true;

#ifdef DEBUG

Serial.print("Long press- Btn:");

Serial.println(m\_pin.pin\_num);

#endif

}

} else {

if (m\_pin.state == HIGH) {

m\_pin.state = LOW;

m\_long\_press = false;

#ifdef DEBUG

Serial.print("Released- Btn:");

Serial.println(m\_pin.pin\_num);

#endif

}

}

}

Файл heatingelement.h

#ifndef HEATINGELEMENT\_H

#define HEATINGELEMENT\_H

#include <stdint.h>

#include <Arduino.h>

#include "pin.h"

class HeatingElement

{

private:

pin m\_pin;

uint8\_t limiter\_switch\_pin;

uint16\_t m\_required\_temp;

uint32\_t m\_timer;

public:

HeatingElement(const uint8\_t pin\_number, const uint8\_t &lower\_sw\_pin,

bool pin\_state = false );

void HeatingElement::turn\_off();

bool HeatingElement::turn\_on();

bool HeatingElement::checkCurrentState(const uint8\_t &curr\_temp, const uint32\_t &curr\_time);

inline bool HeatingElement::state() { return m\_pin.state; }

inline uint8\_t HeatingElement::pin\_num() { return m\_pin.pin\_num; }

inline uint32\_t HeatingElement::timer() { return m\_timer; }

inline uint16\_t HeatingElement::required\_temp() { return m\_required\_temp; }

inline uint16\_t HeatingElement::setRequired\_temp(uint8\_t temp) { m\_required\_temp = temp; return 0; }

inline uint8\_t HeatingElement::setTimer(uint32\_t mins) { m\_timer =(millis() + static\_cast<uint32\_t>(mins) \* 60 \* 60 \* 1000); return 0; }

};

#endif // HEATINGELEMENT\_H

Файл heatingelement.cpp

#include "heatingelement.h"

HeatingElement::HeatingElement(const uint8\_t pin\_number,

const uint8\_t &lower\_sw\_pin,

bool pin\_state = false):

m\_pin{pin\_number, pin\_state},

limiter\_switch\_pin{lower\_sw\_pin},

m\_required\_temp{0},

m\_timer{0}

{

}

bool HeatingElement::turn\_on() {

if (digitalRead(limiter\_switch\_pin) == HIGH) {

m\_pin.state = true;

m\_required\_temp = 40;

return 1;

}

return 0;

}

void HeatingElement::turn\_off() {

m\_pin.state = false;

m\_required\_temp = 0;

m\_timer = 0;

}

bool HeatingElement::checkCurrentState(const uint8\_t &curr\_temp,

const uint32\_t &curr\_time) {

if (m\_pin.state) {

if ((m\_timer && (m\_timer < curr\_time)) ||

(m\_required\_temp <= curr\_temp)) {

turn\_off();

#ifdef DEBUG

Serial.println("heating element turned off");

#endif

return 0;

}

return 1;

}

return 0;

}

#include "heatingelement.h"

HeatingElement::HeatingElement(const uint8\_t pin\_number,

const uint8\_t &lower\_sw\_pin,

bool pin\_state = false):

m\_pin{pin\_number, pin\_state},

limiter\_switch\_pin{lower\_sw\_pin},

m\_required\_temp{0},

m\_timer{0}

{

}

bool HeatingElement::turn\_on() {

if (digitalRead(limiter\_switch\_pin) == HIGH) {

m\_pin.state = true;

m\_required\_temp = 40;

return 1;

}

return 0;

}

void HeatingElement::turn\_off() {

m\_pin.state = false;

m\_required\_temp = 0;

m\_timer = 0;

}

bool HeatingElement::checkCurrentState(const uint8\_t &curr\_temp,

const uint32\_t &curr\_time) {

if (m\_pin.state) {

if ((m\_timer && (m\_timer < curr\_time)) ||

(m\_required\_temp <= curr\_temp)) {

turn\_off();

#ifdef DEBUG

Serial.println("heating element turned off");

#endif

return 0;

}

return 1;

}

return 0;

}

Файл pin.h

#ifndef PIN\_H

#define PIN\_H

struct pin {

uint8\_t pin\_num;

bool state;

};

#endif // PIN\_H

#define NDEBUG

#if !defined(NDEBUG)

#define DEBUG

#endif

Файл boiling\_system.ino

#include <LiquidCrystal\_I2C.h>

#include <OneWire.h>

#include "pin.h"

#include "button.h"

#include "heatingelement.h"

#define ANALOG\_INP\_PIN A0

LiquidCrystal\_I2C lcd(0x27, 16, 2);

OneWire temp\_sensor(2);

uint32\_t main\_timer = 0, menu\_timer = 0;

uint16\_t analog\_input = 0;

const uint8\_t lower\_switch\_pin = 4, higher\_switch\_pin = 5;

Button btn\_Up = {8, LOW}, btn\_Down = {9, LOW};

Button btn\_Cancel = {10, LOW};

HeatingElement heating\_element = {7, lower\_switch\_pin, false};

uint8\_t degree[8] = {

0b00110,

0b01001,

0b01001,

0b00110,

0b00000,

0b00000,

0b00000,

0b00000

};

/\*

\* 0 - show system status; user did not pressed btn\_Up for that yet

\* or rejected last confirmation request

\* 1 - show confirmation menu for changing heating element state

\* or parameters (after confirm heater is on, set default

\* heating\_element.m\_required\_temp=40, m\_timer=0 (does not set)

\* 2 - show confirmation menu for setting requered temp(one click)

\* or timer(double click) on the btn\_Up

\* 3 - show temp setting menu (changing value by potentiometer)

\* 4 - show timer setting menu (changing value by potentiometer)

\* 5 - force abort the heating, flush parameters

\* 6 - show warning: probably heating is not working - after 1 hour

\* curr\_temp is not changed positively (does not emplemented yet)

\*/

enum menuStage {

main\_screen,

confirm\_heating\_params\_change,

heating\_turn\_on\_and\_setup,

temp\_change,

timer\_change,

heating\_turn\_off,

warning,

};

menuStage curr\_menu = main\_screen;

void setup(void) {

Serial.begin(9600);

lcd.init ();

lcd.backlight();

lcd.createChar(0, degree);

pinMode(lower\_switch\_pin, INPUT);

pinMode(higher\_switch\_pin, INPUT);

pinMode(btn\_Up.pin\_num(), INPUT);

pinMode(btn\_Down.pin\_num(), INPUT);

pinMode(btn\_Cancel.pin\_num(), INPUT);

pinMode(heating\_element.pin\_num(), OUTPUT);

lcd.print("starting...");

delay(200);

}

void loop(void) {

uint8\_t data[12];

uint8\_t addr[8];

float curr\_temp;

if (millis() - main\_timer >= 3) {

if (!temp\_sensor.search(addr)) {

//Serial.print("No more addresses.\n");

temp\_sensor.reset\_search();

return;

}

temp\_sensor.reset();

temp\_sensor.select(addr);

temp\_sensor.write(0x44, 1);

// we might do a temp\_sensor.depower() here, but the reset will take care of it.

temp\_sensor.reset();

temp\_sensor.select(addr);

temp\_sensor.write(0xBE); // Read Scratchpad

for (uint8\_t i = 0; i < 9; i++) { // we need 9 bytes

data[i] = temp\_sensor.read();

}

// Convert the data to actual temperature, result is a 16bit signed int

int16\_t raw = (data[1] << 8) | data[0];

byte cfg = (data[4] & 0x60);

if (cfg == 0x00) raw = raw & ~7;

else if (cfg == 0x20) raw = raw & ~3;

else if (cfg == 0x40) raw = raw & ~1;

curr\_temp = (float)raw / 16.0;

if (heating\_element.checkCurrentState(curr\_temp, main\_timer)) {

digitalWrite(heating\_element.pin\_num(), HIGH);

} else {

digitalWrite(heating\_element.pin\_num(), LOW);

}

btn\_Up.checkState();

btn\_Down.checkState();

btn\_Cancel.checkState();

do {

main\_timer += 3;

if (main\_timer < 3)

break; // uint32\_t overflow

} while (main\_timer < (millis() - 3));

}

if (millis() - menu\_timer >= 5) {

lcd.clear();

if (btn\_Cancel.state()) {

if (btn\_Cancel.is\_long\_press()) {

curr\_menu = heating\_turn\_off;

heating\_element.turn\_off();

}

else

curr\_menu = main\_screen;

}

switch (curr\_menu) {

case (main\_screen):

lcd.setCursor(0, 0);

lcd.print(curr\_temp);

lcd.setCursor(4, 0);

lcd.write(0);

lcd.setCursor(5, 0);

lcd.print("C");

lcd.setCursor(0, 1);

lcd.print("Heater:");

lcd.setCursor(7, 1);

if (heating\_element.state())

lcd.print("on");

else

lcd.print("off");

// float switches are connected through pull-down resistor (R=10kOhm)

lcd.setCursor(12, 0);

if (digitalRead(lower\_switch\_pin) == HIGH) // closed

lcd.print(">");

else // open

lcd.print("<");

lcd.setCursor(13, 0);

lcd.print("min");

lcd.setCursor(12, 1);

if (digitalRead(higher\_switch\_pin) == HIGH) // closed

lcd.print(" ");

else // open

lcd.print("<");

if (btn\_Up.state() || btn\_Down.state()) {

curr\_menu = confirm\_heating\_params\_change;

}

lcd.setCursor(13, 1);

lcd.print("max");

break;

case (confirm\_heating\_params\_change):

lcd.setCursor(0,0);

lcd.print("Proceed?");

lcd.setCursor(0, 1);

lcd.print("Up/Down-Yes|Cancel-No");

if (btn\_Up.state() || btn\_Down.state()) {

curr\_menu = heating\_turn\_on\_and\_setup;

}

break;

case (heating\_turn\_on\_and\_setup):

heating\_element.turn\_on();

lcd.setCursor(0,0);

lcd.print("press up/down for");

lcd.setCursor(0,1);

lcd.print("temp/timer change");

if (btn\_Up.state())

curr\_menu = temp\_change;

else if (btn\_Down.state())

curr\_menu = timer\_change;

else if (btn\_Cancel.state())

curr\_menu = main\_screen;

break;

case (temp\_change):

analog\_input = analogRead(ANALOG\_INP\_PIN);

analog\_input = map(analog\_input, 0, 1023, 15, 55);

analog\_input = constrain(analog\_input, curr\_temp, 55);

lcd.setCursor(0, 0);

lcd.print("Set the temp");

lcd.setCursor(0, 1);

lcd.print(analog\_input);

lcd.setCursor(3, 1);

lcd.write(0);

lcd.setCursor(4, 1);

lcd.print("C");

if (btn\_Up.state() || btn\_Down.state()) {

heating\_element.setRequired\_temp(analog\_input);

curr\_menu = main\_screen;

}

break;

case (timer\_change):

analog\_input = analogRead(ANALOG\_INP\_PIN);

analog\_input = map(analog\_input, 0, 1023, 0, 270);

analog\_input = constrain(analog\_input, 0, 270);

uint8\_t mins, hours;

hours = (analog\_input > 60) ? hours = analog\_input / 60 : 0;

mins = analog\_input - hours \* 60;

lcd.setCursor(0, 0);

lcd.print("Set the timer");

lcd.setCursor(0, 1);

lcd.print(hours);

lcd.setCursor(2, 1);

lcd.print("hours");

lcd.setCursor(8, 1);

lcd.print(mins);

lcd.print("min");

if (btn\_Up.state() || btn\_Down.state()) {

heating\_element.setTimer();

curr\_menu = main\_screen;

}

break;

default:

curr\_menu = main\_screen;

break;

}

menu\_timer = millis();

}

}