```
/*
  პროგრამული კოდი, "Arduino UNO R3" დაფაზე არსებული მიკროკონტროლერისთვის
*/

#include "RTClib.h"// საათის მოდულის (RTC) ბიბლიოთეკის დამატება

#define INTERRUPT_PIN 2// მუდმივა, სადაც ინახება იმ პინის ნომერი, რომელთანაც დაკავშირებულია RTC მოდულის SQW პინი და მიკროკონტროლერის წყვეტა (interrupt)...

// 7-სეგმენტა ინდიკატორების კოდურად მართვადი მიკროსქემის (4055) ციფრული პინები
#define b1 A0// ორობითი რიცხვის პირველი ციფრი...
#define b2 A1// ორობითი რიცხვის მეორე ციფრი...
#define b3 A2// ორობითი რიცხვის მესამე ციფრი...
#define b4 A3// ორობითი რიცხვის მეოთხე ციფრი...

#define CLK 7// 4017 მიკროსქემის (მთვლელი) clock პინის შესაბამისი პინის ნომერი
#define DP 8// 7-სეგმენტა ინდიკატორის წერტილოვანი შუქდიოდის შესაბამისი პინის ნომერი
// წანაცვლების რეგისტრებისთვის განკუთვნილი პინები
#define shiftData 4// საინფორმაციო პინი...
#define shiftClock 12// საათის სიხშირის პინი...
#define shiftLatch 13// ჩამკეტი პინი...

// RGB შუქდიოდისთვის განკუთვნილი პინების ნომრები
#define RGB_RED 11// წითელი
#define RGB_GREEN 10// მწვანე
#define RGB_BLUE 9// ლურჯი
```

#define transistorPin 3// P არხიანი ველის ტრანსზისტორის (IRF5210) ჩამკეტისთვის (gate) განკუთვნილი პინის ნომერი

#define redPin 5// მე-2 RGB შუქდიოდის წითელი შუქდიოდის ფეხისთვის განკუთვნილი პინის ნომერი...

#define bluePin 6// მე-2 RGB შუქდიოდის ლურჯი შუქდიოდის ფეხისთვის განკუთვნილი პინის ნომერი...

// გარკვეული სიმბოლოების შესაბამისი ორობითი რიცხვები, 8x8 შუქდიოდურ წერტილოვან მატრიცაზე გამოსატანად...

#define digit_0 {0b00111100, 0b01100110, 0b01100110, 0b01100110, 0b01100110, 0b01100110, 0b01100110, 0b00111100}// 0

#define digit_1 {0b00011000, 0b00111000, 0b00011000, 0b00011000, 0b00011000, 0b00011000, 0b00011000, 0b00111100}// 1

#define digit_2 {0b00000000, 0b00011000, 0b00100100, 0b00000100, 0b00001000, 0b00010000, 0b00111100, 0b00000000}// 2

#define char_space {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000}// " "

#define char_hyphen {0b00000000, 0b00000000, 0b00000000, 0b01111110, 0b01111110, 0b00000000, 0b00000000, 0b00000000}// "-"

#define letter_E {0b01111100, 0b01000000, 0b01000000, 0b01111100, 0b01000000, 0b01000000, 0b01000000, 0b01111100}// "E"

#define geo_a {0b00010000, 0b00010000, 0b00001000, 0b00000100, 0b00000010, 0b01000010, 0b01000010, 0b00111100}// "ა"

#define geo_e {0b00011000, 0b00100100, 0b00000100, 0b00000100, 0b00000100, 0b00000100, 0b00100100, 0b00011000}// "ე"

#define geo_i {0b00111100, 0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b00100100}// "ი"

#define geo_l {0b00011000, 0b00100100, 0b00100000, 0b00100000, 0b00010000, 0b00001000, 0b00000100, 0b00000100}// "ლ"

#define geo_x {0b00100000, 0b00100000, 0b00100000, 0b00100000, 0b00111000, 0b00100100, 0b00100100, 0b00011000}// "ხ"

#define RTC_SQW_PIN_MODE DS1307_SquareWave1HZ// მუდმივა, სადაც ინახება საათის მოდულის (RTC) SQW პინის რეჟიმი

```
// RTC ბიბლიოთეკის ობიექტები...
RTC_DS1307 rtc;
DateTime now;

volatile boolean timeChanged;// ცვლადი, რომელიც განკუთვნილია იმის დასადგენად, შეიცვალა თუ არა დრო (RTC)...

// დროის შესაანახი ცვლადები
byte second;
byte minute;
byte hour;

// თარიღის შესანახი ცვლადები
byte day;
byte month;
int year;

byte displayIndex;// 7-სეგმენტა ინდიკატორის რიგობრივი ნომრის შესანახი ცვლადი
byte displayDigits[6];// მასივი, რომელიც ინახავს 7-სეგმენტა ინდიკატორებზე გამოსატან ციფრებს
boolean displayDots[6] = {0, 1, 0, 1, 0, 0};// მასივი, რომელიც ინახავს 7-სეგმენტა ინდიკატორების წერტილოვანი შუქდიოდების ლოგიკურ მდგომარეობებს
// ცვლადები, რომლებიც ინახავენ მიმდინარე დროს მილიწამებში
unsigned long t;
unsigned long t2;
unsigned long t3;
unsigned long t4;
unsigned long t5;
```

```
unsigned long t6;
unsigned long t7;

int RGB_VAL;// "oneValueRGB" ფუნქციისთვის განკუთვნილი გლობალური ცვლადი
boolean RGB_VAL_DIRECTION = 1;// შუქდიოდის ფერის ცვლილების მიმართულება...
boolean userInputTimeExpired;
boolean dateScrollEnabled;
boolean countDown;
boolean BIDIRECTIONAL_RGB_ENABLED;
byte BIDIRECTIONAL_RGB_DELAY;
boolean digitZeroDissapearEnabled;
boolean newYearScrollEnabled;
boolean snowEnabled;
byte snowMode;
int RGB_VAL1;
int RGB_VAL2;
byte SOFT_RGB_DELAY;
boolean softRandomRGBenabled;
boolean RGBrandomDelayEnabled;
byte RGB_BYTE_VALUE;
byte blinkRGBmode;
byte rand_val1;
byte rand_val2;
byte rand_val3;
boolean blinkColoursRGBenabled;
byte lineIndex;
boolean lineDirection;
boolean lineMode;
boolean matrixLineRandomIndex;
```

```
boolean matrixLinesEnabled;
boolean matrixSquareRandomIndex;
boolean matrixSquaresEnabled;
boolean matrixDotRandom;
boolean matrixDotsEnabled;
byte matrixDotNumber;
byte dotIndex;
byte dotIndex2 = 4;
boolean fallingDotsEnabled;
boolean jumpScrollDirection;
byte jumpScrollIndex;
boolean matrixSmileEnabled;
boolean matrixJumpingSmileEnabled;
byte lastSecond;
boolean matrixScrollSmileEnabled;
boolean matrix_UGLIMES_enabled;
boolean changeModeInJumpingSymbolFunction;
byte matrixRowIndex;
boolean fullMatrixScrollEnabled;
boolean blinkMicrosRGBenabled;
boolean randomRGBmode;
boolean softRGBmode;
byte countRGBmodes;
boolean whiteColourEnable;
byte RGB_COLOUR_TRANSITION_VALUE;
byte GLOBAL_RANDOM_RED_BLUE_CHANGE_DELAY_TIME;
boolean randomRedBlueEnabled;

/*
```

ცვლადი, რომლის სიდიდის მიხედვით განისაზღვრება შუქდიოდურ წერტილოვან მატრიცაზე გამოტანილი გამოსახულების პოზიცია

მას შეუძლია შეინახოს ლოგიკური 1 ან ლოგიკური 0

ლოგიკური 0-ს და ლოგიკური 1-ს შესაბამისი გამოსახულებები, ერთმანეთისგან 180°-ით განსხვავდება

*/

```
boolean reverseMatrix = 0;


String received;

boolean numberScrolling;


// წარწერა "UG-LiMES"-ის შემადგენელი სიმბოლოების შესაბამისი ორობითი რიცხვები, შუქდიოდურ წერტილოვან მატრიცაზე გამოსატანად

byte limes[10][8] = {

  char_space,// " "

  {0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b00111100},// U

  {0b00111100, 0b01000010, 0b01000000, 0b01000000, 0b01011110, 0b01000010, 0b01000010, 0b00111110},// G

  char_hyphen,// -

  {0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b01111110},// L

  {0b00000000, 0b00010000, 0b00000000, 0b00010000, 0b00010000, 0b00010000, 0b00010000, 0b00010000},// i

  {0b01000010, 0b01100110, 0b01011010, 0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b01000010},// M

  letter_E,// E

  {0b00011000, 0b00100100, 0b00100000, 0b00010000, 0b00001000, 0b00000100, 0b00100100, 0b00011000},// S

  char_space,// " "

};
```

// სიმბოლო "🙂"-ს (ღიმილის) შესაბამისი ორობითი რიცხვები, შუქდიოდურ წერტილოვან მატრიცაზე გამოსატანად

```
byte smile[8] = {
  0b00111100, 0b01000010, 0b10100101, 0b10000001, 0b10100101, 0b10011001, 0b01000010, 0b00111100
};
```

// წარწერა "იუჯი-ლიმესი"-ის შემადგენელი სიმბოლოების შესაბამისი ორობითი რიცხვები, შუქდიოდურ წერტილოვან მატრიცაზე გამოსატანად

```
byte limes_geo[13][8] = {
  char_space,// " "
  geo_i,// ი
  {0b00110110, 0b01001010, 0b00000010, 0b00000010, 0b00000010, 0b00000010, 0b01000100, 0b00111000},// უ
  {0b00001000, 0b00011000, 0b00001100, 0b00001000, 0b00000100, 0b00000100, 0b00100100, 0b00011000},// ჯ
  geo_i,// ი
  char_hyphen,// -
  geo_l,// ლ
  geo_i,// ი
  {0b00011000, 0b00100100, 0b00000100, 0b00011100, 0b00100100, 0b00100100, 0b00100100, 0b00011000},// მ
  geo_e,// ე
  {0b00100000, 0b00100000, 0b00100000, 0b00100000, 0b00101000, 0b00100100, 0b00100100, 0b00011000},// ს
  geo_i,// ი
  char_space// " "
};
```

// uglimes სურათის მსგავსი გამოსახულების 8x8 შუქდიოდურ წერტილოვან მატრიცაზე გამოსატანი რიცხვებისთვის განკუთვნილი 8 ელემენტიანი მასივი

```
byte uglimes_icon[8] = {
  0b00111100, 0b00100100, 0b00011000, 0b00111100, 0b01000010, 0b10100101, 0b10000001, 0b01111110
};

// 2 განზომილებიანი მასივი, სადაც ინახება შუქდიოდურ წერტილოვან მატრიცაზე გამოსატანი ათობითი ციფრების შესაბამისი ორობითი რიცხვები
byte digits[10][8] = {
  digit_0,// 0
  digit_1,// 1
  digit_2,// 2
  {0b00000000, 0b00111100, 0b00000100, 0b00000100, 0b00111100, 0b00000100, 0b00000100, 0b00111100},// 3
  {0b00000000, 0b00100100, 0b00100100, 0b00100100, 0b00111100, 0b00000100, 0b00000100, 0b00000100},// 4
  {0b00111100, 0b00100000, 0b00100000, 0b00111000, 0b00000100, 0b00000100, 0b00100100, 0b00011000},// 5
  {0b00011000, 0b00100100, 0b00100000, 0b00111000, 0b00100100, 0b00100100, 0b00100100, 0b00011000},// 6
  {0b00111100, 0b00000100, 0b00000100, 0b00000100, 0b00000100, 0b00000100, 0b00000100, 0b00000100},// 7
  {0b00000000, 0b00011000, 0b00100100, 0b00100100, 0b00011000, 0b00100100, 0b00100100, 0b00011000},// 8
  {0b00011000, 0b00100100, 0b00100100, 0b00011100, 0b00000100, 0b00000100, 0b00100100, 0b00011000}// 9
};

// 2 განზომილებიანი მასივი, რომელიც განკუთვნილია თარიღის (დღე, თვე, წელი) შესანახად
byte date[12][8] = {
  char_space,// " "
  char_space,// დღე (ათეულები)
  char_space,// დღე (ერთეულები)
```

```
  char_hyphen,// სიმბოლო "-"
  char_space,// თვე (ათეულები)
  char_space,// თვე (ერთეულები)
  char_hyphen,// სიმბოლო "-"
  char_space,// წელი (ათასეულები)
  char_space,// წელი (ასეულები)
  char_space,// წელი (ათეულები)
  char_space,// წელი (ერთეულები)
  char_space// " "
};

byte newYear[33][8] = {
  char_space,// " "
  geo_a,// "ა"
  geo_x,// "ხ"
  geo_a,// "ა"
  geo_l,// "ლ"
  geo_i,// "ი"
  char_space,// " "
  {0b00110110, 0b00101000, 0b00100000, 0b00100000, 0b00111000, 0b00100100, 0b00100100,
0b00011000},// "წ"
  geo_e,// "ე"
  geo_l,// "ლ"
  geo_i,// "ი"
  char_space,// " "
  digit_2,// 2
  digit_0,// 0
  digit_2,// 2
```

```
  digit_1,// 1
  char_space,// " "
  char_hyphen,// "-"
  char_space,// " "
  {0b00000000, 0b01000010, 0b01100010, 0b01010010, 0b01001010, 0b01000110, 0b01000010, 0b00000000},// "N"
  letter_E,// "E"
  {0b01000010, 0b01000010, 0b01000010, 0b01000010, 0b01011010, 0b01011010, 0b01100110, 0b01000010},// "W"
  char_space,// " "
  {0b01000010, 0b00100100, 0b00011000, 0b00011000, 0b00011000, 0b00011000, 0b00011000, 0b00011000},// "Y"
  letter_E,// "E"
  {0b00111100, 0b01000010, 0b01000010, 0b01111110, 0b01000010, 0b01000010, 0b01000010, 0b01000010},// "A"
  {0b01111100, 0b01000010, 0b01000010, 0b01111100, 0b01000010, 0b01000010, 0b01000010, 0b01000010},// "R"
  char_space,// " "
  digit_2,// 2
  digit_0,// 0
  digit_2,// 2
  digit_1,// 1
  char_space// " "
};
```

// 2 განზომილებიანი მასივი, სადაც ინახება თოვლის ფიფქების მსგავსი გამოსახულებების შესაბამისი ორობითი კოდები

```
byte snowflake[11][8] = {
  {0b00111000, 0b01010100, 0b10010010, 0b11101110, 0b10010010, 0b01010100, 0b00111000, 0b00000000},
```

```
  {0b00010000, 0b01010100, 0b00111000, 0b11101110, 0b00111000, 0b01010100, 0b00010000, 0b00000000},

  {0b00101000, 0b01010100, 0b10111010, 0b01101100, 0b10111010, 0b01010100, 0b00101000, 0b00000000},

  {0b10010010, 0b00101000, 0b01111100, 0b10101010, 0b01111100, 0b00101000, 0b10010010, 0b00000000},

  {0b00010000, 0b01101100, 0b01010100, 0b10111010, 0b01010100, 0b01101100, 0b00010000, 0b00000000},

  {0b00111000, 0b00010000, 0b10111010, 0b11101110, 0b10111010, 0b00010000, 0b00111000, 0b00000000},

  {0b10111010, 0b01101100, 0b11000110, 0b10010010, 0b11000110, 0b01101100, 0b10111010, 0b00000000},

  {0b10000010, 0b00101000, 0b01101100, 0b00010000, 0b01101100, 0b00101000, 0b10000010, 0b00000000},

  {0b01000100, 0b10101010, 0b01101100, 0b00010000, 0b01101100, 0b10101010, 0b01000100, 0b00000000},

  {0b00101000, 0b01000100, 0b10101010, 0b00010000, 0b10101010, 0b01000100, 0b00101000, 0b00000000},

  {0b00011000, 0b01011010, 0b00100100, 0b11011011, 0b11011011, 0b00100100, 0b01011010, 0b00011000}

};


byte randomNumbers[8] = {2, 6, 0, 5, 3, 7, 4, 1};// "შემთხვევითობის" პრინციპით წინასწარ შერჩეული ციფრები


// ფუნქცია, რომელიც განკუთვნილია RTC მოდულიდან დროს და თარიღის შესაბამის ცვლადებში შესანახად

void getDateTime()  {

  now = rtc.now();

  // დროის შენახვა

  second = now.second();

  minute = now.minute();

  hour = now.hour();
```

```
  // თარიღის შენახვა
  day = now.day();
  month = now.month();
  year = now.year();
}

// 7-სეგმენტა ინდიკატორებზე, გამოსახულებების გამოსახვის ფუნქცია
void sevenSegments(byte digit, boolean DP_STATE)  {
  digitalWrite(b1, digit & (1 << 0));
  digitalWrite(b2, digit & (1 << 1));
  digitalWrite(b3, digit & (1 << 2));
  digitalWrite(b4, digit & (1 << 3));
  digitalWrite(DP, !DP_STATE);
}

// ფუნქცია, რომელიც თითოეულ 7-სეგმენტაზე გამოსახავს შესაბამის გამოსახულებებს
void multiplexSegments()  {
  sevenSegments(0b1111, 0);// 7-სეგმენტა ინდიკატორების ჩაქრობა
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
  sevenSegments(displayDigits[displayIndex], displayDots[displayIndex]);
  if (++displayIndex == 6)  {
    displayIndex = 0;
  }
}

// ფუნქცია, რომელსაც გადაყავს დროის შესაბამისი რიცხვები, 7-სეგმენტაზე გამოსატან ციფრებად
void setTime7segments()  {
```

```
  displayDigits[0] = hour / 10;// საათის ათეულები
  displayDigits[1] = hour % 10;// საათის ერთეულები
  displayDigits[2] = minute / 10;// წუთის ათეულები
  displayDigits[3] = minute % 10;// წუთის ერთეულები
  displayDigits[4] = second / 10;// წამის ათეულები
  displayDigits[5] = second % 10;// წამის ერთეულები
}

// შუქდიოდურ წერტილოვან მატრიცასთან დაკავშირებული წანაცვლების რეგისტრების სამართავად განკუთვნილი ფუნქცია
void shiftMatrix(byte rows, byte columns, boolean bitOrder_row, boolean bitOrder_column) {
  if (reverseMatrix)  {
    // თუ "reverseMatrix" ცვლადი ინახავს ლოგიკურ 1-ს, შუქდიოდურ წერტილოვანი მატრიცის გამოსახულება ამობრუნდება 180°-ით...
    bitOrder_row = !bitOrder_row;
    bitOrder_column = !bitOrder_column;
  }
  digitalWrite(shiftLatch, 0);
  shiftOut(shiftData, shiftClock, bitOrder_row, rows);// სტრიქონების სამართავად განკუთვნილი წანაცვლების რეგისტრისთვის ინფორაციის გაგზავნა
  shiftOut(shiftData, shiftClock, bitOrder_column, columns);// სვეტების სამართავად განკუთვნილი წანაცვლების რეგისტრისთვის ინფორაციის გაგზავნა
  digitalWrite(shiftLatch, 1);
}

// შუქდიოდური წერტილოვანი მატრიცის სრულად ანთების ან სრულად ჩაქრობის ფუნქცია (ფუნქციას პარამეტრად თუ გადაეცა ლოგიკური 1, მატრიცა აინთება, ხოლო თუ ლოგიკური 0 - ჩაქრება)
void fullMatrix(boolean matrixState) {
  shiftMatrix(0xFF * matrixState, 0, LSBFIRST, LSBFIRST);
```

```
}


// შუქდიოდურ წერტილოვან მატრიცაზე, x და y კოორდინატების მიხედვით, ერთი შუქდიოდის ასანთებად განკუთვნილი ფუნქცია
void dotMatrix(byte x, byte y)  {
  /*
    ფუნქციის მეორე პარამეტრად გადმოცემული რიცხვით ხდება ციფრი 1-ს ბიტური წანაცვლება მარჯვნიდან მარცხნივ, მიღებული რიცხვი იგზავნება სტრიქონების სამართავად განკუთვნილ წანაცვლების რეგისტრთან
    პირველ პარამეტრად გადმოცემული რიცვითაც იგივე ხდება, თუმცა მიღებული რიცხვის ბიტურად შებრუნებული რიცხვი იგზავნება სვეტების სამართავად განკუთვნილ წანაცვლების რეგისტრთან
  */
  shiftMatrix(1 << y, ~(1 << x), LSBFIRST, LSBFIRST);
}


// ფუნქცია, რომელიც შუქდიოდურ წერტილოვან მატრიცაზე აანთებს მისთვის პირველ პარამეტრად გადაცემული რიცხვის ინდექსის მქონე სტრიქონს, მეორე პარამეტრის მიხედვით
void rowMatrix(byte index, byte number)  {
  shiftMatrix(1 << index, ~number, LSBFIRST, MSBFIRST);
}


// ფუნქცია, რომელიც შუქდიოდურ წერტილოვან მატრიცაზე აანთებს მისთვის პირველ პარამეტრად გადაცემული რიცხვის ინდექსის მქონე სვეტს, მეორე პარამეტრის მიხედვით
void columnMatrix(byte index, byte number)  {
  shiftMatrix(number, ~(1 << index), MSBFIRST, LSBFIRST);
}


// დაყოვნების ფუნქცია
void delayFunction(unsigned int delayTime)  {
  /*
```

```
    პროგრამის მსვლელობის დაყოვნება, ამ ფუნციისთვის პარამეტრად გამდოცემული დროით

    თუმცა დაყოვნების პერიოდში, "loopFunction" ფუნქცია აგრძელებს მუშაობას

  */

  for (t2 = millis(); millis() - t2 <= delayTime; loopFunction());

}


// დაყოვნების ფუნქცია (მიკროწამები)

void delayMicrosecondsFunction(unsigned int delayTime)  {

  /*

    პროგრამის მსვლელობის დაყოვნება, ამ ფუნციისთვის პარამეტრად გამდოცემული დროით

    თუმცა დაყოვნების პერიოდში, "loopFunction" ფუნქცია აგრძელებს მუშაობას

  */

  for (t2 = micros(); micros() - t2 <= delayTime; loopFunction());

}


// ფუნქცია, რომელიც შუქდიოდურ წერტილოვან მატრიცაზე, სტრიქონ-სტრიქონ გამოსახავს მისთვის პარამეტრად გადაცემულ მასივში არსებული რიცხვების შესაბამის გამოსახულებას

void symbolMatrix(byte rows[]) {

  for (byte row = 0; row < 8; row++) {

    rowMatrix(row, rows[row]);

  }

}


// ფუნქცია, რომელიც ტექსტის შემადგენელ სიმბოლოებს შუქდიოდურ წერტილოვან მატრიცაზე მარჯვნიდან მარცხნივ ამოძრავებს

void scrollText(byte chars[][8], byte charactersNumber, byte scrollDelayTime) {

  for (byte charIndex = 0; charIndex < (charactersNumber - 1); charIndex++) {// for ციკლი, რომელიც გადაუყვება თითოეულ სიმბოლოს შესაბამის ორობით რიცხვებს
```

```
   for (byte scroll = 1; scroll <= 8; scroll++) {
    for (t3 = millis(); millis() - t3 < scrollDelayTime; loopFunction())  {
     if (userInputTimeExpired && numberScrolling)  {
      fullMatrix(0);
      break;
     }
     for (byte row = 0; row < 8; row++) {
      rowMatrix(row, (chars[charIndex][row] << scroll) | (chars[charIndex + 1][row] >> (8 - scroll)));
     }
    }
   }
  }
}
```

// ფუნქცია, რომელიც მისთვის პირველ პარამეტრად გადაცემულ ორ განზომილებიან მასივში არსებული რიცხვების შესაბამის გამოსახულებებს, შუქდიოდურ წერტილოვან მატრიცაზე, ზემოდან ქვემოთ ამოძრავებს...

```
void scrollCharsFromTop(byte chars[][8], byte charactersNumber, byte scrollDelayTime, byte fromCorner/* 0 - ზემოდან; 1 - მარცხენა კუთხიდან; 2 - მარჯვენა კუთხიდან */) {
  boolean mixedDirectionMode = 0;
  for (byte charIndex = 0; charIndex < charactersNumber; charIndex++) {// for ციკლი, რომელიც გადაუყვება თითოეულ სიმბოლოს შესაბამის ოროზით რიცხვებს
   if (fromCorner == 3)  {
    mixedDirectionMode = 1;
   }
   if (mixedDirectionMode)  {
    if (++fromCorner > 2)  {
     fromCorner = 0;
    }
   }
```

```
// ზემოდან შუაში
for (byte scroll = 1; scroll <= 8; scroll++) {
  for (t3 = millis(); millis() - t3 < scrollDelayTime; loopFunction())  {
    for (byte row = 0; row < 8; row++) {
      if (row < scroll)  {
        switch (fromCorner)  {
          case 0:
            rowMatrix(row, chars[charIndex][(scroll - 1) - row]);// ზემოდან
            break;
          case 1:
            rowMatrix(row, chars[charIndex][(scroll - 1) - row] << (8 - scroll));// მარცხენა კუთხიდან
            break;
          case 2:
            rowMatrix(row, chars[charIndex][(scroll - 1) - row] >> (8 - scroll));// მარჯვენა კუთხიდან
            break;
        }
      }
      else  {
        rowMatrix(row, 0);
      }
    }
  }
}
// შუიდან ქვემოთ
for (byte scroll = 2; scroll <= 8; scroll++) {
  for (t3 = millis(); millis() - t3 < scrollDelayTime; loopFunction())  {
    for (byte row = 0; row < 8; row++) {
      switch (fromCorner)  {
```

```
          case 0:
            rowMatrix(row + scroll, chars[charIndex][row]);// ზემოდან
            break;
          case 1:
            rowMatrix(row + scroll, chars[charIndex][row] >> (scroll - 1));// მარცხენა კუთხიდან
            break;
          case 2:
            rowMatrix(row + scroll, chars[charIndex][row] << (scroll - 1));// მარჯვენა კუთხიდან
            break;
        }
       }
     }
   }
 }
```

// ფუნქცია, რომელიც ერთ სიმბოლოს შუქდიოდურ წერტილოვან მატრიცაზე მარჯვნიდან მარცხნივ ამოძრავებს

```
void scrollSingle(byte rows[8], byte scrollDelayTime) {
  for (byte scroll = 8; scroll > 0; scroll--) {
    for (t3 = millis(); millis() - t3 < scrollDelayTime; loopFunction())  {
      for (byte row = 0; row < 8; row++) {
        rowMatrix(row, rows[row] >> scroll);
      }
    }
  }
  for (byte scroll = 0; scroll <= 8; scroll++) {
    for (t3 = millis(); millis() - t3 < scrollDelayTime; loopFunction())  {
      for (byte row = 0; row < 8; row++) {
```

```
      rowMatrix(row, rows[row] << scroll);
    }
   }
  }
}

// შუქდიოდურ წერტილოვან მატრიცაზე ციფრების გამოსახვის ფუნქცია
void matrixDigit(byte digit)  {
  for (byte row = 0; row < 8; row++) {
    rowMatrix(row, digits[digit][row]);
  }
}

// შუქდიოდურ წერტილოვან მატრიცაზე, ციფრი 0-ის "შემთხვევითობის" პრინციპით გაუჩინარების ფუნქცია
void randomDissapearDigitZero()  {
  byte rows[8] = digit_0;
  for (byte index = 0; index < 8; index++) {
    rows[randomNumbers[index]] = rows[randomNumbers[index]] & random(0, 256);
    for (t3 = millis(); millis() - t3 < 100; loopFunction())  {
      for (byte row = 0; row < 8; row++) {
        rowMatrix(row, rows[row]);
      }
    }
    rows[randomNumbers[index]] = 0;
  }
  fullMatrix(0);
  delayFunction(100);
}
```

// ფუნქცია, რომელიც მისთვის პარამეტრად გადაცემულ ციფრებს შუქდიოდურ წერტილოვან მატრიცაზე მარჯვნიდან მარცხნივ ამოძრავებს

```
void scrollDigits(String receivedDigits, byte SCROLL_DELAY_TIME) {
  byte receivedDigitsLength = receivedDigits.length();
  byte digitsArray[receivedDigitsLength + 2][8];
  for (byte row_index = 0; row_index < 8; row_index++) {
    digitsArray[0][row_index] = 0;
  }
  for (byte row_index = 0; row_index < 8; row_index++) {
    digitsArray[receivedDigitsLength + 1][row_index] = 0;
  }
  for (byte digitIndex = 0; digitIndex < receivedDigitsLength; digitIndex++) {
    for (byte row_index = 0; row_index < 8; row_index++) {
      digitsArray[digitIndex + 1][row_index] =
digits[String(receivedDigits.charAt(digitIndex)).toInt()][row_index];
    }
  }
  scrollText(digitsArray, receivedDigitsLength + 2, SCROLL_DELAY_TIME);
}
```

// ფუნქცია, რომელიც მიღებულ დროს და თარიღს აქცევს შესაბამის საზღვრებში, რადგან არასწორად გამოგზავნილი ფორმატის შემთხვევაში, მოხდეს მისი შესწორებული ვარიანტის შენახვა...

// მაგალითად: "24:60:99 2019-13-32" ჩასწორდება როგორც: "23:59:59 2019-12-31"; "0:0:0 2020-0-0" -> "0:0:0 2020-1-1"; "0:0:0 2000-1-1" -> "0:0:0 2001-1-1"; "0:0:0 2100-1-1" -> "0:0:0 2099-1-1"...

// შენიშვნა: პროექტში გამოყენებული RTC მოდული ვერ ინახავს 2000 წელზე ნაკლებს და 2099 წელზე მეტს...

```
void sanitizeDateTimeInput()  {
  hour = constrain(hour, 0, 59);
```

```
  minute = constrain(minute, 0, 59);
  second = constrain(second, 0, 59);
  year = constrain(year, 2000, 2099);
  month = constrain(month, 1, 12);
  day = constrain(day, 0, 31);
}

// დროს და თარიღის დაყენების ფუნქცია, "datetime" String პარამეტრის მიხედვით...
void setDateTime(String datetime)  {
  hour = datetime.substring(0, datetime.indexOf(':')).toInt();
  minute = datetime.substring(datetime.indexOf(':') + 1, datetime.indexOf(':', datetime.indexOf(':') + 1)).toInt();
  second = datetime.substring(datetime.indexOf(':', datetime.indexOf(':') + 1) + 1, datetime.indexOf(' ')).toInt();
  year = datetime.substring(datetime.indexOf(' ') + 1, datetime.indexOf('-')).toInt();
  month = datetime.substring(datetime.indexOf('-') + 1, datetime.indexOf('-', datetime.indexOf('-') + 1)).toInt();
  day = datetime.substring(datetime.indexOf('-', datetime.indexOf('-') + 1) + 1).toInt();
  sanitizeDateTimeInput();
  rtc.adjust(DateTime(year, month, day, hour, minute, second));
}

// ფუნქცია, სადაც ხდება UART (Serial) კავშირის მართვა...
void UART_FUNCTION()  {
  if (Serial.available() && !numberScrolling)  {
    received = Serial.readString();
    if (received.charAt(0) == 'n' && !userInputTimeExpired)  {
      numberScrolling = 1;
      scrollDigits(received.substring(1), 100);
      numberScrolling = 0;
```

```
    }
    else if (received.charAt(0) == 't') {
      setDateTime(received.substring(1));
    }
  }
  else if (millis() - t4 >= 1000)  {
    Serial.print("n");
    t4 = millis();
  }
}

// RGB შუქდიოდის მართვის ფუნქცია
void RGB(byte red, byte green, byte blue)  {// ფუნქციის პარამეტრები, რომლებიც შეინახავს RGB შუქდიოდის შემადგენელი ფერების (წითელი, მწვანე, ლურჯი) ინტენსივობის შესაბამის რიცხვებს
  analogWrite(RGB_RED, red);
  analogWrite(RGB_GREEN, green);
  analogWrite(RGB_BLUE, blue);
}

// "RGB_VAL" გლობალურ ცვლადში არსებული რიცხვის მიხედვით, RGB შუქდიოდის შესაბამისი ფერით განათების ფუნქცია
void oneValueRGB()  {
  // ჩამქრალი - წითელი
  if ((RGB_VAL >= 0) && (RGB_VAL < (256)))  {
    RGB((RGB_VAL % 256), 0, 0);
  }
  // წითელი - ყვითელი
  else if ((RGB_VAL >= (256)) && (RGB_VAL < (256 * 2)))  {
```

```
  RGB(0xFF, (RGB_VAL % 256), 0);
}
// ყვითელი - მწვანე
else if ((RGB_VAL >= (256 * 2)) && (RGB_VAL < (256 * 3)))  {
  RGB(0xFF - (RGB_VAL % 256), 0xFF, 0);
}
// მწვანე - ცისფერი
else if ((RGB_VAL >= (256 * 3)) && (RGB_VAL < (256 * 4)))  {
  RGB(0, 0xFF, (RGB_VAL % 256));
}
// ცისფერი - ლურჯი
else if ((RGB_VAL >= (256 * 4)) && (RGB_VAL < (256 * 5)))  {
  RGB(0, 0xFF - (RGB_VAL % 256), 0xFF);
}
// ლურჯი - იისფერი
else if ((RGB_VAL >= (256 * 5)) && (RGB_VAL < (256 * 6)))  {
  RGB((RGB_VAL % 256), 0, 0xFF);
}
// იისფერი - წითელი
else if ((RGB_VAL >= (256 * 6)) && (RGB_VAL < (256 * 7)))  {
  RGB(0xFF, 0, 0xFF - (RGB_VAL % 256));
}
// წითელი - თეთრი
else if ((RGB_VAL >= (256 * 7)) && (RGB_VAL < (256 * 8)))  {
  RGB(0xFF, RGB_VAL % 256, RGB_VAL % 256);
}
// თეთრი - ჩამქრალი
else if ((RGB_VAL >= (256 * 8)) && (RGB_VAL < (256 * 9)))  {
```

```
    RGB(0xFF - (RGB_VAL % 256), 0xFF - (RGB_VAL % 256), 0xFF - (RGB_VAL % 256));
  }
}
```

// ფუნქცია, რომელიც ცვლის შუქდიოდის ფერს ხან ერთი, ხან მეორე მიმართულებით, მისთვის პარამეტრად გადაცემულ დროში (მილიწამი), ერთი ერთეულით

```
void softBidirectionalRGB(byte changeDelay) {
  if (millis() - t5 >= changeDelay)  {
    if (RGB_VAL < 0)  {
      RGB_VAL_DIRECTION = 1;
    }
    else if (RGB_VAL > (256 * 9)) {
      RGB_VAL_DIRECTION = 0;
    }
    if (RGB_VAL_DIRECTION)  {
      RGB_VAL++;
    }
    else  {
      RGB_VAL--;
    }
    oneValueRGB();
    t5 = millis();
  }
}

void softRandomRGBsetup(int val1, int val2) {
  RGB_VAL1 = val1;
  RGB_VAL2 = val2;
  if (RGB_VAL1 > RGB_VAL2)  {
```

```
    int temp = RGB_VAL1;
    RGB_VAL1 = RGB_VAL2;
    RGB_VAL2 = temp;
  }
  RGB_VAL = RGB_VAL1;
}

void softRandomRGBloop(byte local_delayMillis)  {
  if (millis() - t5 >= local_delayMillis)  {
    if (++RGB_VAL < RGB_VAL2)  {
      oneValueRGB();
    }
    else  {
      softRandomRGBsetup(RGB_VAL2, random(0, 256 * 9));
    }
    t5 = millis();
  }
}

void blinkColoursRGB(byte local_delayMillis)  {
  if (millis() - t5 >= local_delayMillis)  {
    if (RGB_BYTE_VALUE == 0)  {
      RGB_VAL_DIRECTION = 1;
      if (++blinkRGBmode > 7)  {
        blinkRGBmode = 0;
        rand_val1 = random(1, 256);
        rand_val2 = random(1, 256);
        rand_val3 = random(1, 256);
      }
```

```
  }
  else if (RGB_BYTE_VALUE == 255) {
   RGB_VAL_DIRECTION = 0;
  }
  if (RGB_VAL_DIRECTION)  {
   RGB_BYTE_VALUE++;
  }
  else  {
   RGB_BYTE_VALUE--;
  }
  switch (blinkRGBmode)  {
    case 0:
     RGB(constrain(RGB_BYTE_VALUE, 0, rand_val1), constrain(RGB_BYTE_VALUE, 0, rand_val2),
constrain(RGB_BYTE_VALUE, 0, rand_val3));
     break;
    case 1:
     RGB(RGB_BYTE_VALUE, 0, 0);
     break;
    case 2:
     RGB(RGB_BYTE_VALUE, RGB_BYTE_VALUE, 0);
     break;
    case 3:
     RGB(0, RGB_BYTE_VALUE, 0);
     break;
    case 4:
     RGB(0, RGB_BYTE_VALUE, RGB_BYTE_VALUE);
     break;
    case 5:
     RGB(0, 0, RGB_BYTE_VALUE);
```

```
      break;
     case 6:
      RGB(RGB_BYTE_VALUE, 0, RGB_BYTE_VALUE);
      break;
     case 7:
      RGB(RGB_BYTE_VALUE, RGB_BYTE_VALUE, RGB_BYTE_VALUE);
      break;
   }
   t5 = millis();
  }
}

// RGB შუქდიოდის ფერის ნელი ცვლილების ფუნქცია...
void RGBcoloursTransition(unsigned long delayTime) {
  if (millis() - t5 >= delayTime) {
   // წითელი-ყვითელი
   if (countRGBmodes == 0)  {
    RGB(255, RGB_COLOUR_TRANSITION_VALUE, 0);
   }
   // ყვითელი-მწვანე
   else if (countRGBmodes == 1) {
    RGB(255 - RGB_COLOUR_TRANSITION_VALUE, 255, 0);
   }
   // მწვანე-ცისფერი
   else if (countRGBmodes == 2) {
    RGB(0, 255, RGB_COLOUR_TRANSITION_VALUE);
   }
   // ცისფერი-ლურჯი
```

```
  else if (countRGBmodes == 3) {
    RGB(0, 255 - RGB_COLOUR_TRANSITION_VALUE, 255);
  }
  // ლურჯი-იისფერი
  else if (countRGBmodes == 4) {
    RGB(RGB_COLOUR_TRANSITION_VALUE, 0, 255);
  }
  // იისფერი-წითელი
  else if (countRGBmodes == 5) {
    RGB(255, 0, 255 - RGB_COLOUR_TRANSITION_VALUE);
  }
  else if (whiteColourEnable) {
    // წითელი-თეთრი
    if (countRGBmodes == 6)  {
      RGB(255, RGB_COLOUR_TRANSITION_VALUE, RGB_COLOUR_TRANSITION_VALUE);
    }
    // თეთრი-ყვითელი
    else  {
      RGB(255, 255, 255 - RGB_COLOUR_TRANSITION_VALUE);
    }
  }
  if (++RGB_COLOUR_TRANSITION_VALUE == 0)  {
    ++countRGBmodes;// იზრდება "countRGBmodes" ცლადის სიდიდე 1 ერთეულით
    if (countRGBmodes == 6)  {
      whiteColourEnable = !whiteColourEnable;// "whiteColourEnable" ცვლადის ლოგიკური
შებრუნება
    }
    if ((countRGBmodes > 5) && !whiteColourEnable)  {
      countRGBmodes = 0;
```

```
    }
    else if (countRGBmodes > 7)  {
      countRGBmodes = 1;
    }
   }
   t5 = millis();
 }
}
```

// ფუნქცია, რომელიც დააბრუნებს ლოგიკურ 1-ს იმ შემთხვევაში, თუ მისთვის პარამეტრებად გადაცემული დრო და თარიღი უდრის მიმდინარე დროს და თარიღს.. წინააღმდეგ შემთხვევაში, დააბრუნებს ლოგიკურ 0-ს..

```
boolean datetime_equals(byte local_second, byte local_minute, byte local_hour, byte local_day, byte local_month, int local_year) {
  return ((second == local_second) && (minute == local_minute) && (hour == local_hour) && (day == local_day) && (month == local_month) && (year == local_year));
}
```

// თარიღის მასივის ციფრებით შევსების ფუნქცია, მისთვის გადაცემული პარამეტრების მიხედვით

```
void fillDateArrayWithDigitsArray(byte dateArrayIndex, byte digitsArrayIndex) {
  for (byte local_index = 0; local_index < 8; local_index++) {
    date[dateArrayIndex][local_index] = digits[digitsArrayIndex][local_index];
  }
}
```

// ფუნქცია, რომელიც მისთვის პარამეტრებად გადაცემულ თარიღს (დღე, თვე, წელი) გამოიტანს შუქდიოდურ წერტილოვან მატრიცაზე, მარჯვნიდან მარცხნივ.. გამოსახულების დაყოვნების დრო (მილიწამი), ფუნქციას "local_scrollDelayTime" პარამეტრად გადაეცემა..

```
void scrollDate(byte local_day, byte local_month, int local_year, byte local_scrollDelayTime) {
  fillDateArrayWithDigitsArray(1, local_day / 10);// დღე (პირველი ციფრი)
```

```
  fillDateArrayWithDigitsArray(2, local_day % 10);// დღე (მეორე ციფრი)
  fillDateArrayWithDigitsArray(4, local_month / 10);// თვე (პირველი ციფრი)
  fillDateArrayWithDigitsArray(5, local_month % 10);// თვე (მეორე ციფრი)
  fillDateArrayWithDigitsArray(7, local_year / 1000);// წელი (პირველი ციფრი)
  fillDateArrayWithDigitsArray(8, (local_year / 100) % 10);// წელი (მეორე ციფრი)
  fillDateArrayWithDigitsArray(9, (local_year / 10) % 10);// წელი (მესამე ციფრი)
  fillDateArrayWithDigitsArray(10, local_year % 10);// წელი (მეოთხე ციფრი)
  scrollText(date, 12, local_scrollDelayTime);// "scrollText" ფუნქციის გამოძახება...
}

void bidirectionalMatrixLine(byte minIndex, byte maxIndex)  {
  if (lineIndex == minIndex)  {
    lineDirection = 1;
  }
  else if (lineIndex == maxIndex) {
    lineDirection = 0;
  }
  if (lineDirection)  {
    lineIndex++;
  }
  else  {
    lineIndex--;
  }
}

void matrixLines()  {
  if (millis() - t6 >= 100)  {
    if (!matrixLineRandomIndex)  {
```

```
      bidirectionalMatrixLine(0, 7);
    }
    else  {
      lineIndex = random(0, 8);
    }
    if (!lineMode)  {
      rowMatrix(lineIndex, 0xFF);
    }
    else  {
      columnMatrix(lineIndex, 0xFF);
    }
    t6 = millis();
  }
}

void matrixSquares() {
  if (millis() - t6 >= 100)  {
    if (!matrixSquareRandomIndex)  {
      bidirectionalMatrixLine(0, 3);
    }
    else  {
      lineIndex = random(0, 4);
    }
    t6 = millis();
  }
  rowMatrix(lineIndex, (0xFF << lineIndex) & (0xFF >> lineIndex));
  columnMatrix(lineIndex, (0xFF << lineIndex) & (0xFF >> lineIndex));
  rowMatrix(7 - lineIndex, (0xFF << lineIndex) & (0xFF >> lineIndex));
  columnMatrix(7 - lineIndex, (0xFF << lineIndex) & (0xFF >> lineIndex));
```

```
}

void matrixDots() {
  if (millis() - t6 >= 25)  {
    if (!matrixDotRandom)  {
      if (++matrixDotNumber > 63)  {
        matrixDotNumber = 0;
      }
      if ((matrixDotNumber / 8) % 2)
        dotMatrix(7 - (matrixDotNumber % 8), matrixDotNumber / 8);
      else
        dotMatrix(matrixDotNumber % 8, matrixDotNumber / 8);
    }
    else  {
      dotMatrix(random(0, 8), random(0, 8));
    }
    t6 = millis();
  }
}

void matrixFallingDotsOnInclinedPlane() {
  if (millis() - t6 >= 100)  {
    if (++dotIndex > 8)  {
      dotIndex = 0;
    }
    if (++dotIndex2 > 8)  {
      dotIndex2 = 0;
    }
    t6 = millis();
```

```
  }
  bidirectionalMatrixLine(0, 7);
  dotMatrix(dotIndex + 1, dotIndex);
  dotMatrix(dotIndex2 + 1, dotIndex2);
  rowMatrix(lineIndex, 0xFF << 7 - lineIndex);
}

void matrixJumpSymbol(byte chars[], byte scrollDelayTime) {
  if (jumpScrollIndex == 0)  {
    jumpScrollDirection = 1;
  }
  else if (jumpScrollIndex == 8) {
    jumpScrollDirection = 0;
  }
  if (jumpScrollDirection)  {
    jumpScrollIndex++;
  }
  else  {
    jumpScrollIndex--;
  }
  for (t3 = millis(); millis() - t3 < scrollDelayTime; loopFunction())  {
    for (byte row = 0; row < 8; row++) {
      if (row < jumpScrollIndex)  {
        rowMatrix(row, chars[7 - ((jumpScrollIndex - 1) - row)]);
      }
      else  {
        rowMatrix(row, 0);
      }
    }
```

```
  }
}

void fullMatrixScroll() {
  if (millis() - t6 >= 75)  {
    if (lineIndex > 0) {
      lineIndex--;
      fullMatrix(0);
    }
    t6 = millis();
  }
  for (matrixRowIndex = 0; matrixRowIndex < lineIndex; matrixRowIndex++) {
    rowMatrix(matrixRowIndex, 0xFF);
  }
}

void randomRedBlue(byte changeDelayTime)  {
  if (millis() - t7 >= changeDelayTime)  {
    analogWrite(redPin, random(0, 256));
    analogWrite(bluePin, random(0, 256));
    t7 = millis();
  }
}

// ფუნქცია, სადაც დროს და თარიღის მიხედვით ხდება ელექტრული სქემის მართვა...
void functionDateTime() {
  if (datetime_equals(0, 59, 23, 31, 12, 2020))  {// თუ დრო და თარიღი არის 2020-12-31 23:59:00
    userInputTimeExpired = 1;// უქმდება შუქდიოდურ წერტილოვან მატრიცაზე გამოსატანი
რიცხვების NodeMCU-დან მიღება
```

```
}
else if (datetime_equals(1, 59, 23, 31, 12, 2020))  {
  dateScrollEnabled = 1;
}
else if (datetime_equals(30, 59, 23, 31, 12, 2020)) {
  dateScrollEnabled = 0;
}
else if (datetime_equals(51, 59, 23, 31, 12, 2020)) {
  countDown = 1;
}
else if (datetime_equals(0, 0, 0, 1, 1, 2021)) {
  randomSeed(now.unixtime());
  BIDIRECTIONAL_RGB_DELAY = 10;
  BIDIRECTIONAL_RGB_ENABLED = 1;
}
else if (datetime_equals(1, 0, 0, 1, 1, 2021) && countDown) {
  countDown = 0;
  digitZeroDissapearEnabled = 1;
}
else if (datetime_equals(2, 0, 0, 1, 1, 2021)) {
  dateScrollEnabled = 1;
}
else if (datetime_equals(15, 0, 0, 1, 1, 2021)) {
  dateScrollEnabled = 0;
  newYearScrollEnabled = 1;
}
else if (datetime_equals(0, 1, 0, 1, 1, 2021)) {
  BIDIRECTIONAL_RGB_DELAY = 1;
  newYearScrollEnabled = 0;
```

```
  matrixLinesEnabled = 1;
}
else if (datetime_equals(15, 1, 0, 1, 1, 2021)) {
  lineMode = 1;
}
else if (datetime_equals(23, 1, 0, 1, 1, 2021)) {
  lineMode = 0;
  matrixLineRandomIndex = 1;
}
else if (datetime_equals(30, 1, 0, 1, 1, 2021)) {
  BIDIRECTIONAL_RGB_ENABLED = 0;
  softRandomRGBsetup(random(0, 256 * 9), random(0, 256 * 9));
  SOFT_RGB_DELAY = 1;
  softRandomRGBenabled = 1;
}
else if (datetime_equals(31, 1, 0, 1, 1, 2021)) {
  lineMode = 1;
}
else if (datetime_equals(39, 1, 0, 1, 1, 2021)) {
  matrixLinesEnabled = 0;
  matrixSquaresEnabled = 1;
}
else if (datetime_equals(45, 1, 0, 1, 1, 2021)) {
  RGBrandomDelayEnabled = 1;
}
else if (datetime_equals(47, 1, 0, 1, 1, 2021)) {
  matrixSquareRandomIndex = 1;
}
else if (datetime_equals(55, 1, 0, 1, 1, 2021)) {
```

```
  matrixSquaresEnabled = 0;
  matrixDotsEnabled = 1;
}
else if (datetime_equals(0, 2, 0, 1, 1, 2021)) {
  softRandomRGBenabled = 0;
  RGB_VAL_DIRECTION = 1;
  SOFT_RGB_DELAY = 4;
  blinkColoursRGBenabled = 1;
}
else if (datetime_equals(3, 2, 0, 1, 1, 2021)) {
  matrixDotRandom = 1;
}
else if (datetime_equals(11, 2, 0, 1, 1, 2021)) {
  matrixDotsEnabled = 0;
  fallingDotsEnabled = 1;
}
else if (datetime_equals(19, 2, 0, 1, 1, 2021)) {
  fallingDotsEnabled = 0;
  snowEnabled = 1;
}
else if (datetime_equals(30, 2, 0, 1, 1, 2021)) {
  SOFT_RGB_DELAY = 1;
}
else if (datetime_equals(34, 2, 0, 1, 1, 2021)) {
  snowMode = 1;// თოვლის ფიფქები მარცხნიდან...
}
else if (datetime_equals(45, 2, 0, 1, 1, 2021)) {
  blinkColoursRGBenabled = 0;
  randomRGBmode = 1;
```

```
}
else if (datetime_equals(49, 2, 0, 1, 1, 2021)) {
  snowMode = 2;// თოვლის ფიფქები მარჯვნიდან...
}
else if (datetime_equals(0, 3, 0, 1, 1, 2021)) {
  randomRGBmode = 0;
  blinkMicrosRGBenabled = 1;
}
else if (datetime_equals(4, 3, 0, 1, 1, 2021)) {
  snowMode = 3;// თოვლის ფიფქები შერეული მიმართულებებიდან...
}
else if (datetime_equals(15, 3, 0, 1, 1, 2021)) {
  GLOBAL_RANDOM_RED_BLUE_CHANGE_DELAY_TIME = 200;
  randomRedBlueEnabled = 1;
}
else if (datetime_equals(25, 3, 0, 1, 1, 2021)) {
  digitalWrite(transistorPin, 0);
}
else if (datetime_equals(40, 3, 0, 1, 1, 2021)) {
  GLOBAL_RANDOM_RED_BLUE_CHANGE_DELAY_TIME = 100;
}
else if (datetime_equals(55, 3, 0, 1, 1, 2021)) {
  GLOBAL_RANDOM_RED_BLUE_CHANGE_DELAY_TIME = 50;
}
else if (datetime_equals(10, 4, 0, 1, 1, 2021)) {
  GLOBAL_RANDOM_RED_BLUE_CHANGE_DELAY_TIME = 25;
}
else if (datetime_equals(25, 4, 0, 1, 1, 2021)) {
  GLOBAL_RANDOM_RED_BLUE_CHANGE_DELAY_TIME = 10;
```

```
  }
  else if (datetime_equals(30, 4, 0, 1, 1, 2021)) {
    GLOBAL_RANDOM_RED_BLUE_CHANGE_DELAY_TIME = 250;
  }
  else if (datetime_equals(31, 4, 0, 1, 1, 2021)) {
    snowEnabled = 0;
    matrixSmileEnabled = 1;
  }
  else if (datetime_equals(42, 4, 0, 1, 1, 2021)) {
    matrixSmileEnabled = 0;
    jumpScrollIndex = 8;
    matrixJumpingSmileEnabled = 1;
  }
  else if (datetime_equals(52, 4, 0, 1, 1, 2021)) {
    blinkMicrosRGBenabled = 0;
    matrixJumpingSmileEnabled = 0;
    fullMatrix(1);
  }
  else if (datetime_equals(53, 4, 0, 1, 1, 2021)) {
    lineIndex = 8;
    fullMatrixScrollEnabled = 1;
  }
  else if (datetime_equals(54, 4, 0, 1, 1, 2021)) {
    fullMatrixScrollEnabled = 0;
    matrixJumpingSmileEnabled = 0;
    softRGBmode = 1;
    matrix_UGLIMES_enabled = 1;
  }
}
```

```
// ფუნქცია, სადაც წერია ბრძანებები, რომლებიც უწყვეტად სრულდება...
void loopFunction() {
  // ყოველ 1 წამში, ხდება დროის და თარიღის მიღება RTC მოდულიდან და დროის შესაბამისი ციფრები ინახება "displayDigits" მასივში, 7-სეგმენტა ინდიკატორებისთვის
  if (timeChanged)  {
    getDateTime();
    setTime7segments();
    Serial.print('t' + String(hour) + ':' + String(minute) + ':' + String(second) + ' ' + String(year) + '-' + String(month) + '-' + String(day));
    timeChanged = 0;
  }

  if (millis() - t >= 1)  {// ამ ფიგურულ ფრჩხილებში ჩაწერილი ბრძანებები შესრულდება ყოველ 1 მილიწამში
    multiplexSegments();// "multiplexSegments" ფუნქციის გამოძახება
    t = millis();
  }
  if (lastSecond != second)  {
    functionDateTime();
    lastSecond = second;
  }
  UART_FUNCTION();
  if (BIDIRECTIONAL_RGB_ENABLED)  {
    softBidirectionalRGB(BIDIRECTIONAL_RGB_DELAY);
  }
  else if (softRandomRGBenabled) {
    if (RGBrandomDelayEnabled)  {
      SOFT_RGB_DELAY = random(1, 25);
```

```
  }
  softRandomRGBloop(SOFT_RGB_DELAY);
 }
 else if (blinkColoursRGBenabled) {
  blinkColoursRGB(SOFT_RGB_DELAY);
 }
 else if (blinkMicrosRGBenabled) {
  RGB_VAL = (micros() / 500) % (256 * 9);
  oneValueRGB();
 }
 else if (randomRGBmode) {
  if (millis() - t5 >= 100)  {
   RGB(random(0, 256), random(0, 256), random(0, 256));
   t5 = millis();
  }
 }
 else if (fullMatrixScrollEnabled) {
  RGB_VAL = (micros() / 250) % (256 * 9);
  oneValueRGB();
 }
 else if (softRGBmode) {
  RGBcoloursTransition(10);
 }
 if (randomRedBlueEnabled)  {
  randomRedBlue(GLOBAL_RANDOM_RED_BLUE_CHANGE_DELAY_TIME);
 }
}

// წყვეტის (interrupt) ფუნქცია...
```

```
void timeChanged_ISR()  {
  timeChanged = 1;
}

void setup()  {
  Serial.begin(115200);// UART კავშირის დაწყება 115200kb/s (1000 ბიტი წამში) სიჩქარით
  Serial.setTimeout(2);// UART კავშირით ინფორმაციის მიღებისას, ლოდინის დრო იქნება 2 მილიწამი (0.002 წამი).. ეს წესი არ ეხება ისეთ შემთხვევებს, როცა ხდება მხოლოდ 1 სიმბოლოს ამოკითხვა (მაგალითად Serial.read() ბრძანება)..
  rtc.begin();// RTC მოდულსა და მიკროკონტროლერს შორის კავშირის დაწყება
  // 4055 მიკროსქემის შესაბამისი პინების გამოსავალ პინებად გამოცხადება
  pinMode(b1, OUTPUT);
  pinMode(b2, OUTPUT);
  pinMode(b3, OUTPUT);
  pinMode(b4, OUTPUT);

  // RGB შუქდიოდის შესაბამისი პინების გამოსავალ პინებად გამოცხადება
  pinMode(RGB_RED, OUTPUT);
  pinMode(RGB_GREEN, OUTPUT);
  pinMode(RGB_BLUE, OUTPUT);

  pinMode(CLK, OUTPUT);// 4017 მიკროსქემის (მთვლელი) clock პინის შესაბამისი პინის გამოსავალ პინად გამოცხადება
  pinMode(DP, OUTPUT);// 7-სეგმენტა ინდიკატორის წერტილოვანი შუქდიოდის შესაბამისი პინის გამოსავალ პინად გამოცხადება
  pinMode(shiftData, OUTPUT);// წანაცვლების რეგისტრის საინფორმაციო პინის შესაბამისი პინის გამოსავალ პინად გამოცხადება
  pinMode(shiftClock, OUTPUT);// წანაცვლების რეგისტრის საათის სიხშირის პინის შესაბამისი პინის გამოსავალ პინად გამოცხადება
```

```
  pinMode(shiftLatch, OUTPUT);// წანაცვლების რეგისტრის ჩამკეტი პინის შესაბამისი პინის გამოსავალ პინად გამოცხადება

  pinMode(transistorPin, OUTPUT);

  pinMode(redPin, OUTPUT);

  pinMode(bluePin, OUTPUT);

  digitalWrite(transistorPin, 1);// P არხიანი ტრანზისტორის ჩამკეტზე ლოგიკური 1-ის გაგზავნა, რადგან თავდაპირველად სქემა იყოს გამორთული...

  for (byte count = 0; count < 5; count++) {// მთვლელის clock ფეხზე იმპულსების გაგზავნა, რადგან პროგრამული კოდის მსვლელობის დაწყებისას, პირველი 7-სეგმენტა იყოს ანთებული

    digitalWrite(CLK, 1);

    digitalWrite(CLK, 0);

  }

  fullMatrix(0);// შუქდიოდურ წერტილოვანი მატრიცის სრულად ჩაქრობა

  // RTC საათის მოდულის SQW პინის რეჟიმის დაყენება "RTC_SQW_PIN_MODE" მუდმივის მიხედვით, თუ ამჟამინდელი რეჟიმი არ არის აღნიშნულ მუდმივაში არსებული რეჟიმი

  if (rtc.readSqwPinMode() != RTC_SQW_PIN_MODE)  {

    rtc.writeSqwPinMode(RTC_SQW_PIN_MODE);

  }

  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), timeChanged_ISR, RISING);// წყვეტა (interrupt) აღიძვრება და შესრულდება "timeChanged_ISR" ფუნქცია, როცა "INTERRUPT_PIN" მუდმივაში არსებული რიცხვის მქონე ნომრის ციფრულ პინზე მოხდება ცვლილება ლოგიკური 0-დან ლოგიკურ 1-ზე

}


void loop() {

  loopFunction();// "loopFunction" ფუნქციის გამოძახება

  if (dateScrollEnabled)  {

    scrollDate(day, month, year, 100);

  }

  else if (countDown) {
```

```
  matrixDigit((60 - second) % 10);
}
else if (digitZeroDissapearEnabled) {
  randomDissapearDigitZero();
  digitZeroDissapearEnabled = 0;
}
else if (newYearScrollEnabled) {
  scrollText(newYear, 33, 100);
}
else if (snowEnabled) {
  scrollCharsFromTop(snowflake, 11, 100, snowMode);
}
else if (matrixLinesEnabled) {
  matrixLines();
}
else if (matrixSquaresEnabled) {
  matrixSquares();
}
else if (matrixDotsEnabled) {
  matrixDots();
}
else if (fallingDotsEnabled) {
  matrixFallingDotsOnInclinedPlane();
}
else if (matrixSmileEnabled) {
  symbolMatrix(smile);
}
else if (matrixJumpingSmileEnabled) {
  matrixJumpSymbol(smile, 50);
```

```
  }
  else if (fullMatrixScrollEnabled) {
    fullMatrixScroll();
  }
  else if (matrix_UGLIMES_enabled) {
    scrollSingle(smile, 100);
    scrollText(limes, 10, 100);
    scrollText(limes_geo, 13, 100);
    scrollSingle(uglimes_icon, 100);
  }
}
```