```
/*
 * GccApplication2.c
 *
 * Created: 2019-05-18 15:03:23
 * Author : heb15cpe
 */

#define F_CPU 8000000UL
#define F_SCL 100000UL
#define Prescaler 1
#define TWBR_val ((((F_CPU / F_SCL) / Prescaler) - 16 ) / 2)
#define I2C_READ 0x01
#define I2C_WRITE 0x00
#define RTC_READ            (0xDF)
#define RTC_WRITE           (0xDE)

#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/twi.h>

/** METODER */
void setCommand(char c);
void writeChar(char c);
void displayOn();
void writeText(char String[]);
void writeTwoDigits(uint8_t data);
void clearDisplay();
void keyPressed();
void setKeyboard();
void setDiods();
void dispTime();
void showTempStats();
void alarmCold();
void alarmWarm();
void updateTemps();

void i2c_init(void);
uint8_t i2c_start(uint8_t address);
uint8_t i2c_write(uint8_t data);
uint8_t i2c_read_ack(void);
uint8_t i2c_read_nack(void);
```

```
uint8_t i2c_transmit(uint8_t address, uint8_t* data, uint16_t length);
uint8_t i2c_receive(uint8_t address, uint8_t* data, uint16_t length);
uint8_t i2c_writeReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length);
uint8_t i2c_readReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length);
void i2c_stop(void);

/** ATTRIBUT */
char val;
volatile uint16_t pushed;
volatile uint8_t adc_ready;
volatile uint16_t reading;
volatile int room;

int inOrOut = 1; //för att hålla koll på om vi är inne eller ute
int maxOrMin = 1; //för att hålla koll på om vi styr max eller min
int option = 0; // Parameter för alternativval för pil upp och ned.
int h_config;
int min_config;
int year_config;
int month_config;
int day_config;


/** Temperatur */
double tempIn; //innetemperatur
double tempOut; //utomhustemperatur
int maxTempOut; //högsta temp ute
int minTempOut; //minsta temp ute
int maxTempIn; //högsta temp inne
int minTempIn; //minsta temp inne
int maxLimTempOut = 25; //maxgräns ute
int minLimTempOut = 10; //míngräns ute
int maxLimTempIn = 25; //maxgräns inne
int minLimTempIn = 15; //mingräns inne
double avgIn; //medeltemperatur inne
double avgOut; //medeltemperatur ute

int sumOfTempsIn;
int nbrOfTempsIn;
int sumOfTempsOut;
int nbrOfTempsOut;

uint8_t set_sec=0b00000000, set_min=0b00000000, set_h=0b00000000, set_year =
0b01010101, set_month = 0b01010101, set_day = 0b01010101, get_sec ,get_min ,get_h,
get_year, get_month, get_day;
uint8_t real_sec;
```

```
uint8_t real_min;
uint8_t real_h;
uint8_t real_year;
uint8_t real_month;
uint8_t real_day;


/** Hur flödet går mellan processor och andra komponenter */
void setDataFlow(){
	DDRA=0x00; //A port är bara input, från knappsatsen osv
	DDRB=0xFF; //B port är bara output, till displayen osv
	DDRC=0xC0;
	DDRD=0x1A;
	ADMUX |= (1<<MUX1) | (1<<MUX2) | (1<<REFS0);
	ADCSRA |= (1<<ADEN) | (1<<ADIE) |(1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0) ;
	diodsOff();
}

/** DISPLAY */
/** Sätter commando */
void setCommand(char c) {
	_delay_ms(100);
	PORTB = c;
	PORTD |= (1<< PD1);
	PORTD &= ~(1<< PD4) ;
	PORTD &= ~(1<< PD3) ;
	PORTD &= ~(1<< PD1);
	PORTD |= (1<< PD1);
}

/** Setup för displayen */
void setDisplay(){
	setCommand(0x3C);
}

void clearDisplay(){
	setCommand(0x01);
	_delay_ms(10);
}

void displayOn(){
	setCommand(0x0F);
	clearDisplay();
	_delay_ms(100);
	writeText("Tjofaderittan!");
	_delay_ms(2000);
```

```
}

/**För att kunna skriva tecken*/
void writeChar(char c){
	_delay_us(100);
	PORTB = c;
	PORTD |= (1<< PD1);
	PORTD |= (1<< PD4) ;
	PORTD &= ~(1<< PD3) ;
	PORTD &= ~(1<< PD1);
	PORTD |= (1<< PD1);
}

/** För att skriva strängar genom att loopa chars*/
void writeText(char String[]){
	int i = 0;
	while(String[i] != '\0'){
		writeChar(String[i]);
		i++;
	}
}

/**Skriver int på displayen*/
void writeInt(int h){
	if(h/10 == 0){
		writeChar('0' + h%10);
	}else{
		writeInt(h/10);
		writeInt(h%10);
	}
}

void writeTime(){
	if(real_h < 10) {
		writeInt(0);
		}
		writeInt(real_h);
		writeChar(':');
	if(real_min < 10) {
		writeInt(0);
		}
		writeInt(real_min);
		writeChar(':');
	if(real_sec < 10) {
		writeInt(0);
		}
```

```
		writeInt(real_sec);
}

/** KNAPPSATS */
void setKeyboard() {
	MCUCR = 0b00000011;
	GICR = 0b01000000;
}

/** INTERRUPTION */
ISR(INT0_vect){
	val = PINA;
	pushed =1;
}

/** Vad som ska göras vid knapptryckning. */
void keyPressed(){
	if (val == 0xF0){ //knapp 1: Visa nuvarande temperatur
		clearDisplay();
		writeTime();
		setCommand(0xC0);
		if(inOrOut > 0) {
			writeText("Ute: ");
			writeInt(tempOut);
			} else {
			writeText("Inne: ");
			writeInt(tempIn);
		}
	}
	if (val == 0xF1){ //knapp 2: Byt mellan inne och ute
		clearDisplay();
		writeText("Ute eller inne?");
		setCommand(0xC0);
		inOrOut = -inOrOut;
		if(inOrOut > 0) {
			writeText("Utomhus");
		} else {
			writeText("Inomhus");
		}
	}
	if (val == 0xF2){ //knapp 3: Byt mellan min och max
		clearDisplay();
		option = 0;
		writeText("Min eller max?");
		setCommand(0xC0);
		maxOrMin = -maxOrMin;
```

```
		if(maxOrMin > 0) {
			writeText("Max");
		} else if(maxOrMin < 0) {
			writeText("Min");
		}
}
if (val == 0xF3){ //Höj gräns för nuvarande inställning (max/min + in/out)
		switch(option) {
			case 0 :
		if (inOrOut > 0){
			if (maxOrMin > 0){
				maxLimTempOut++;
			} else {
				minLimTempOut++;
			}
		} else {
			if (maxOrMin > 0){
				maxLimTempIn++;
			} else {
				minLimTempIn++;
			}
		}
		break;
		case 1: h_config++;
		break;
		case 2 : min_config++;
		break;
		case 3 : year_config++;
		break;
		case 4 : month_config++;
		break;
		case 5 : day_config++;
		break;
		}
}
if (val == 0xF4){ //knapp 4: visa temperaturstatistik
		clearDisplay();
		showTempStats();
}
if (val == 0xF5){ //knapp 5: Visa max- & mingränser
		clearDisplay();
		if (inOrOut > 0){
			writeText("MaxL: ");
			writeInt(maxLimTempOut);
			setCommand(0xC0);
			writeText("MinL: ");
```

```
                writeInt(minLimTempOut);
        } else {
                writeText("MaxL: ");
                writeInt(maxLimTempIn);
                setCommand(0xC0);
                writeText("MinL: ");
                writeInt(minLimTempIn);
        }
}
if (val == 0xF6){ //knapp 6:
        option++;
        if(option == 6) {
                option = 0;
        }
        clearDisplay();
        switch(option) {
                case 0 :
                writeText("Temperaturgränser");
                break;
                case 1 :
                writeText("Ställ in: ");
                setCommand(0xC0);
                writeText("timmar");
                break;
                case 2 :
                writeText("Ställ in: ");
                setCommand(0xC0);
                writeText("minuter");
                break;
                case 3 :
                writeText("Ställ in: ");
                setCommand(0xC0);
                writeText("ar");
                break;
                case 4 :
                writeText("Ställ in: ");
                setCommand(0xC0);
                writeText("manad");
                break;
                case 5 :
                writeText("Ställ in: ");
                setCommand(0xC0);
                writeText("dag");
                break;
        }
}
```

```
	if (val == 0xF7){ ////Sänk gräns för nuvarande inställning (max/min + in/out)
		switch(option) {
			case 0:
		if (inOrOut > 0){
			if (maxOrMin > 0){
				maxLimTempOut--;
				} else {
				minLimTempOut--;
			}
			} else {
			if (maxOrMin > 0){
				maxLimTempIn--;
			} else {
				minLimTempIn--;
			}
		}
		break;
		case 1: h_config--;
		break;
		case 2 : min_config--;
		break;
		case 3 : year_config--;
		break;
		case 4 : month_config--;
		break;
		case 5 : day_config--;
		break;
			}
	}
	if (val == 0xFB){ //MENY: Visa startmeny
		clearDisplay();
		displayOn();
	}
	if (val == 0xFE){ //DEL: Stäng av dioder
		diodsOff();
	}
	if (val == 0xFF){ //ENTER: Sätt på dioder
		diodsOn();
	}
	if (val == 0xF8 || val == 0xF9 || val == 0xFA || val == 0xFC || val == 0xFD) { //fel
knappval
		clearDisplay();
		writeText("Tryck pa en");
		setCommand(0xC0);
		writeText("annan knapp");
	}
```

```
}

/** DIODER */
/** Tänder röda dioden. */
void diodsOn(){
	redDiodOn();
	yellowDiodOn();
}

void diodsOff(){
	redDiodOff();
	yellowDiodOff();
}

void redDiodOff() {
	PORTC &= ~(1<<PD7);
}

/** Tänder gula dioden. */
void yellowDiodOff() {
	PORTC &= ~(1<<PD6);
}

/** Släcker röda dioden. */
void redDiodOn() {
	PORTC |= (1<<PD7);
}

/** Släcker gula dioden. */
void yellowDiodOn() {
	PORTC |= (1<<PD6);
}

/** TERMOMETER & TEMPERATUR */
ISR(ADC_vect){
	reading = ADC;
	if (reading > 10){
		if(room == 0) {
			tempIn = (reading * 0.43) - 273;
		} else {
			tempOut = (reading * 0.43) - 273;
		}
		updateTemps();
		adc_toggle();
		ADCSRA |= 1 << ADSC;
	}
```

```
}

/** Toggle ´för avläsning av inomhus respektive utomhus */
void adc_toggle() {
	if( room == 0) {
		room = 1;
		ADMUX |= (1<<MUX0);
	} else {
		room = 0;
		ADMUX &=~ (1<<MUX0);
	}
}

void updateTemps(){
	if (room == 0){
		sumOfTempsIn += tempIn;
		nbrOfTempsIn++;
		if (tempIn > maxTempIn) {
			maxTempIn = tempIn;
		} else if (tempIn < minTempIn){
			minTempIn = tempIn;
		}
		avgIn = sumOfTempsIn / nbrOfTempsIn;
	} else {
		sumOfTempsOut += tempOut;
		nbrOfTempsOut++;
		if (tempOut > maxTempOut) {
			maxTempOut = tempOut;
		} else if (tempOut < minTempOut){
			minTempOut = tempOut;
		}
		avgOut = sumOfTempsOut / nbrOfTempsOut;
	}
}

/** Här kommer implementeras metod för att fram max- och mintemperatur över de senaste
tre dygnen */

/** Visar dygnets minimum och maximum på displayen */
void showTempStats(){
	if(inOrOut > 0) {
		writeText("Max: ");
		writeInt(maxTempOut);
		writeText(" Min: ");
		writeInt(minTempOut);
		setCommand(0xC0);
```

```
                writeText("Medel: ");
                writeInt(avgOut);
        } else {
                writeText("Max: ");
                writeInt(maxTempIn);
                writeText(" Min: ");
                writeInt(minTempIn);
                setCommand(0xC0);
                writeText("Medel: ");
                writeInt(avgIn);
        }
}
/** Hanterar larmfunktionen för ifall det blir för kallt ute eller inne */
void alarmCold(){
        clearDisplay();
        yellowDiodOn();
        if (inOrOut > 0){
                writeText("KALLT UTE!");
        } else {
                writeText("KALLT INNE!");
        }
}
/** Hanterar larmfunktionen för ifall det blir för varmt ute eller inne */
void alarmWarm(){
        clearDisplay();
        redDiodOn();
        if (inOrOut > 0){
                writeText("VARMT UTE!");
        } else {
                writeText("VARMT INNE!");
        }
}

/** Kollar om någon temperaturgräns är passerad */
void checkLimit() {
        if(inOrOut > 0) {
                if(tempOut > maxLimTempOut) {
                        alarmWarm();
                } else if(tempOut < minLimTempOut) {
                        alarmCold();
                }
        } else {
                if(tempIn > maxLimTempIn) {
                        alarmWarm();
                } else if (tempIn < minLimTempIn){
                        alarmCold();
```

```
                }
        }
}

/** RTC */
void i2c_init(void)
{
        TWBR = (uint8_t)TWBR_val;
}

uint8_t i2c_start(uint8_t address)
{
        // reset TWI control register
        TWCR = 0;
        // transmit START condition
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        // wait for end of transmission
        while( !(TWCR & (1<<TWINT)) );

        // check if the start condition was successfully transmitted
        if((TWSR & 0xF8) != TW_START){ return 1; }

        // load slave address into data register
        TWDR = address;
        // start transmission of address
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wait for end of transmission
        while( !(TWCR & (1<<TWINT)) );

        // check if the device has acknowledged the READ / WRITE mode
        uint8_t twst = TW_STATUS & 0xF8;
        if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

        return 0;
}

uint8_t i2c_write(uint8_t data)
{
        // load data into data register
        TWDR = data;
        // start transmission of data
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wait for end of transmission
        while( !(TWCR & (1<<TWINT)) );

        if( (TWSR & 0xF8) != TW_MT_DATA_ACK ){ return 1; }
```

```
	return 0;
}

uint8_t i2c_read_ack(void)
{

	// start TWI module and acknowledge data after reception
	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
	// wait for end of transmission
	while( !(TWCR & (1<<TWINT)) );
	// return received data from TWDR
	return TWDR;
}

uint8_t i2c_read_nack(void)
{

	// start receiving without acknowledging reception
	TWCR = (1<<TWINT) | (1<<TWEN);
	// wait for end of transmission
	while( !(TWCR & (1<<TWINT)) );
	// return received data from TWDR
	return TWDR;
}

uint8_t i2c_transmit(uint8_t address, uint8_t* data, uint16_t length)
{
	if (i2c_start(address | I2C_WRITE)) return 1;

	for (uint16_t i = 0; i < length; i++)
	{
		if (i2c_write(data[i])) return 1;
	}

	i2c_stop();

	return 0;
}

uint8_t i2c_receive(uint8_t address, uint8_t* data, uint16_t length)
{
	if (i2c_start(address | I2C_READ)) return 1;

	for (uint16_t i = 0; i < (length-1); i++)
	{
```

```
                data[i] = i2c_read_ack();
        }
        data[(length-1)] = i2c_read_nack();

        i2c_stop();

        return 0;
}

uint8_t i2c_writeReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length)
{
        if (i2c_start(devaddr | 0x00)) return 1;

        i2c_write(regaddr);

        for (uint16_t i = 0; i < length; i++)
        {
                if (i2c_write(data[i])) return 1;
        }

        i2c_stop();

        return 0;
}

uint8_t i2c_readReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length)
{
        if (i2c_start(devaddr)) return 1;

        i2c_write(regaddr);

        if (i2c_start(devaddr | 0x01)) return 1;

        for (uint16_t i = 0; i < (length-1); i++)
        {
                data[i] = i2c_read_ack();
        }
        data[(length-1)] = i2c_read_nack();

        i2c_stop();

        return 0;
}

void i2c_stop(void)
{
```

```
	// transmit STOP condition
	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
}

void set_clock(void)
{
	i2c_start(RTC_WRITE);
	i2c_write(0x00);
	i2c_write(set_sec | (1 << 7) );
	i2c_write(set_min);
	i2c_write(set_h);
	i2c_stop();
}

void get_time()
{
	i2c_start(RTC_WRITE);
	i2c_write(0x00);
	i2c_start(RTC_READ);
	get_sec = i2c_read_ack();
	real_sec = ((get_sec & 0b01110000) >> 4) * 10 + (get_sec & 0b00001111);
	get_min = i2c_read_ack();
	real_min = ((get_min & 0b01110000) >> 4) * 10 + (get_min & 0b00001111) +
min_config;
	get_h = i2c_read_nack();
	real_h = ((get_h & 0b00110000) >> 4) * 10 + (get_h & 0b00001111) + h_config;
	if(real_h > 23) {
		real_h = 23;
		h_config--;
	}

	if(real_h >= 23 && real_min >= 59 && real_sec >= 59){
		real_h = 0;
		real_min = 0;
		real_sec = 0;
		min_config = 0;
		h_config = 0;
	}
	if(real_min > 59) {
		real_min = 0;
		min_config = 0;
		h_config++;
	}
			i2c_stop();
}
```

```
int main(void){
    /* Replace with your application code */
        setDataFlow();
        setKeyboard();
        setDisplay();
        clearDisplay();
        displayOn();

        i2c_init();

        i2c_start(0xDE);
        i2c_write(0x00);
        i2c_start(0xDF);

        i2c_stop();
        set_clock();

        sei();
        ADCSRA |= 1 << ADSC;
        while (1) {
                get_time();
                //checkLimit();
                if(pushed==1){
                        keyPressed();
                        pushed = 0;
                }
    }
}
```