ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (MICROLAB)

8η Εργαστηριακή Αναφορά στο μάθημα “**ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ”** του 7ου Εξαμήνου

των φοιτητών της **ομάδας 17**,

Εμμανουήλ Αναστάσιου Σερλή, Α.Μ. 03118125
Ευάγγελου Περσόπουλου, Α.Μ.: 03121701

Ζήτημα 8.1:

Στο ζητούμενο ερώτημα, στέλνονται στον server οι κατάλληλες εντολές για σύνδεση στο δίκτυο (σε κατάλληλη διεύθυνση url), ενώ οι απαντήσεις του server απεικονίζονται στην lcd οθόνη. Για τον σκοπό αυτό, χρησιμοποιούνται οι ρουτίνες usart_transmit_string() και usart_receive_string() οι οποίες είναι υπεύθυνες για την αποστολή και την λήψη μηνυμάτων τύπου char και κάνουν χρήση των δοσμένων ρουτίνων usart_transmit() και usart_receive() αντίστοιχα.

Ζήτημα 8.2:

Στο ζητούμενο ερώτημα, επεκτείνεται ο κώδικας του ζητούμενου 8.1 έτσι ώστε να πραγματοποιείται ανανέωση του status του ασθενή ανάλογα με τα keyboard inputs, σε συνδυασμό με τις μετρήσεις θερμοκρασίας και πίεσης. Συγκεκριμένα, πραγματοποιείται ανάγνωση των keyboard inputs για 3sec, μέχρις ώτου πατηθεί κάποιο εκ των “7” (για status = NURSE CALL) ή “#” (για status = OK). Στην συνέχεια, ανεξάρτητα από το αν πατήθηκε κάποιο πλήκτρο ή όχι, πραγματοποιούνται μετρήσεις θερμοκρασίας και πίεσης, ενώ τα συνολικά αποτελέσματα απεικονίζονται στην lcd screen. Αξίζει να σημειωθεί ότι, αν πατηθεί NURSE CALL, τότε το status θα παραμείνει έτσι και για τις επόμενες επαναλήψεις της while loop, μέχρι να πατηθεί η δίεση, για να γίνει το status OK (εκτός αν η θερμοκρασία και η πίεση το αλλάξουν εκ νέου σε CHECK TEMP ή CHECK PRESSURE αντίστοιχα)

Ζήτημα 8.3:

Στο ζητούμενο ερώτημα, επεκτείνεται ο κώδικας του ζητούμενου 8.2 για να πραγματοποιηθεί αποστολή payload στον server. Η κατάλληλη μορφοποίηση και αποστολή του payload γίνεται από την ρουτίνα usart_transmit_payload_all(). Τέλος, στέλνεται η εντολή transmit, με την αντίστοιχη απάντηση του server να λαμβάνεται και να αποτυπώνεται στην lcd οθόνη.

Παρακάτω ακολουθεί ο συνολικός κώδικας σε C για τα 3 ζητούμενα. Εντός της int main(), αναγράφεται σε σχόλια πώς αντιστοιχούν τα κομμάτια κώδικα με καθένα από τα ζητούμενα 8.1-8.3

```
#define F_CPU 16000000UL //running
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include<math.h>
#include<string.h>
#define my_sizeof(type) ((char *)(&type+1)-(char*)(&type))

//lab04 -> lcd routines
void write_2_nibbles(char x)
```

```
{
	char y=PIND & 0x0f;
	char x1=x & 0xf0;
	x1=x1+y;
	PORTD=x1;
	PORTD=PORTD | (1<<PD3);
	PORTD=PORTD & (0<<PD3);
	x=x<<4 | x>>4;
	x=x & 0xf0;
	PORTD=x+y;
	PORTD=PORTD | (1<<PD3);
	PORTD=PORTD & (0<<PD3);
}

void lcd_data(char x)
{
	PORTD=PORTD | (1<<PD2);
	write_2_nibbles(x);
	_delay_us(50);
}

void lcd_command(char x)
{
	PORTD=PORTD | (0<<PD2);
	write_2_nibbles(x);
	_delay_us(50);
}

void lcd_init (void)
{
	_delay_ms(40);
	PORTD=0x30;
	PORTD=PORTD | (1<<PD3);
	PORTD=PORTD & (0<<PD3);
	_delay_us(38);
	PORTD=0x30;
	PORTD=PORTD | (1<<PD3);
	PORTD=PORTD & (0<<PD3);
	_delay_us(38);
	PORTD=0x20;
	PORTD=PORTD | (1<<PD3);
	PORTD=PORTD & (0<<PD3);
	_delay_us(38);
	lcd_command(0x28);
	lcd_command(0x0c);
	lcd_command(0x01);
	_delay_ms(500);
	lcd_command(0x06);
}

//lab05 -> twi routines
#define PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
```

```
#define TWI_READ 1 // reading from twi device
#define TWI_WRITE 0 // writing to twi device
#define SCL_CLOCK 100000L // twi clock in Hz

//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2


// PCA9555 REGISTERS
typedef enum {
 REG_INPUT_0 = 0,
 REG_INPUT_1 = 1,
 REG_OUTPUT_0 = 2,
 REG_OUTPUT_1 = 3,
 REG_POLARITY_INV_0 = 4,
 REG_POLARITY_INV_1 = 5,
 REG_CONFIGURATION_0 = 6,
 REG_CONFIGURATION_1 = 7,
} PCA9555_REGISTERS;

//----------- Master Transmitter/Receiver -------------------
#define TW_START 0x08
#define TW_REP_START 0x10

//---------------- Master Transmitter ----------------------
#define TW_MT_SLA_ACK 0x18
#define TW_MT_SLA_NACK 0x20
#define TW_MT_DATA_ACK 0x28

//---------------- Master Receiver ----------------
#define TW_MR_SLA_ACK 0x40
#define TW_MR_SLA_NACK 0x48
#define TW_MR_DATA_NACK 0x58

#define TW_STATUS_MASK 0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)

//initialize TWI clock
void twi_init(void)
{
 TWSR0 = 0; // PRESCALER_VALUE=1
 TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
}

// Read one byte from the twi device ( request more data from device)
unsigned char twi_readAck(void)
{
TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
while(!(TWCR0 & (1<<TWINT)));
 return TWDR0;
}
```

```
//Read one byte from the twi device, read is followed by a stop
condition
unsigned char twi_readNak(void)
{
TWCR0 = (1<<TWINT) | (1<<TWEN);
while(!(TWCR0 & (1<<TWINT)));

return TWDR0;
}

// Issues a start condition and sends address and transfer direction.
// return 0 = device accessible, 1= failed to access device
unsigned char twi_start(unsigned char address)
{
uint8_t twi_status;

// send START condition
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

// wait until transmission completed
while(!(TWCR0 & (1<<TWINT)));

// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return
1;

// send device address
TWDR0 = address;
TWCR0 = (1<<TWINT) | (1<<TWEN);

// wail until transmission completed and ACK/NACK has been received
while(!(TWCR0 & (1<<TWINT)));

// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if ( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
 {
 return 1;
 }
return 0;
}

// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address)
{
 uint8_t twi_status;
 while ( 1 )
 {
// send START condition
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
```

```
 // wait until transmission completed
 while(!(TWCR0 & (1<<TWINT)));

 // check value of TWI Status Register.
 twi_status = TW_STATUS & 0xF8;
 if ( (twi_status != TW_START) && (twi_status != TW_REP_START))
continue;

 // send device address
 TWDR0 = address;
 TWCR0 = (1<<TWINT) | (1<<TWEN);

 // wail until transmission completed
 while(!(TWCR0 & (1<<TWINT)));

 // check value of TWI Status Register.
 twi_status = TW_STATUS & 0xF8;
 if ( (twi_status == TW_MT_SLA_NACK )||(twi_status ==TW_MR_DATA_NACK)
)
 {
 /* device busy, send stop condition to terminate write operation */
 TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

 // wait until stop condition is executed and bus released
 while(TWCR0 & (1<<TWSTO));

 continue;
 }
break;
 }
}

// Send one byte to twi device, Return 0 if write successful or 1 if
write failed
unsigned char twi_write( unsigned char data )
{
// send data to the previously addressed device
TWDR0 = data;
TWCR0 = (1<<TWINT) | (1<<TWEN);
// wait until transmission completed
while(!(TWCR0 & (1<<TWINT)));
if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
return 0;
}
// Send repeated start condition, address, transfer direction
//Return: 0 device accessible
// 1 failed to access device
unsigned char twi_rep_start(unsigned char address)
{
 return twi_start( address );
}
```

```
// Terminates the data transfer and releases the twi bus
void twi_stop(void)
{
 // send stop condition
TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
// wait until stop condition is executed and bus released
while(TWCR0 & (1<<TWSTO));
}
void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
{
 twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
 twi_write(reg);
 twi_write(value);
 twi_stop();
}
uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
{
uint8_t ret_val;

 twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
 twi_write(reg);
 twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
 ret_val = twi_readNak();
 twi_stop();

 return ret_val;
}

//lab06 -> keypad-scan
unsigned char scan_row(unsigned int row)
{
    PCA9555_0_write(REG_OUTPUT_1, row);
    _delay_ms(2);
    unsigned char value = PCA9555_0_read(REG_INPUT_1);
    return value;
}

unsigned char scan_keypad(void)
{
    unsigned char key=0;
    key = scan_row(0xFE); //check row 1
    if (key != 0xFE) return key;
    key = scan_row(0xFD); //check row 2
    if (key != 0xFD) return key;
    key = scan_row(0xFB); //check row 3
    if (key != 0xFB) return key;
    key = scan_row(0xF7); //check row 4
    return key;
}

unsigned char scan_keypad_rising_edge_2(void)
{
    unsigned char check_1 = scan_keypad();
    _delay_ms(10);
    unsigned char check_2 = scan_keypad();
    while (check_1 == check_2)
    {
        //_delay_ms(20);
        check_2 = scan_keypad();
    }
    return check_1;
}

unsigned char keypad_to_ascii(void)
{
// unsigned char x = scan_keypad_rising_edge_2();
unsigned char x = scan_keypad();
 asm("NOP");
 //1st row
 if (x == 0b11101110) return '*';
 if (x == 0b11011110) return '0';
 if (x == 0b10111110) return '#';
 if (x == 0b01111110) return 'D';
 //2nd row
 if (x == 0b11101101) return '7';
 if (x == 0b11011101) return '8';
 if (x == 0b10111101) return '9';
 if (x == 0b01111101) return 'C';
 //3rd row
 if (x == 0b11101011) return '4';
 if (x == 0b11011011) return '5';
 if (x == 0b10111011) return '6';
 if (x == 0b01111011) return 'B';
 //4th row
 if (x == 0b11100111) return '1';
 if (x == 0b11010111) return '2';
 if (x == 0b10110111) return '3';
 if (x == 0b01110111) return 'A';

 return 0xF7;   //if nothing is pressed
}


//lab07 -> one-wire for temperature measurement
int _sign;  //for sign of temperature
unsigned int one_wire_reset(void)
{
    unsigned int x=0;
    DDRD |= (1<<PD4); //set PD4 as output
    PORTD &= (0<<PD4);
    _delay_us(480);
    DDRD &= (0<<PD4); //set PD4 as input
    PORTD &= (0<<PD4); //disable pull-up
    asm("nop");
    _delay_us(100);
    x = PIND;
    x = x >>4;
    asm("nop");
    _delay_us(380);
    asm("nop");
    if ((x & 0x01) == 0x01) return 0;
    else return 1;   //not connected device
}

unsigned int one_wire_receive_bit (void)
{
    uint8_t x;
    DDRD |= (1<<PD4); //set PD4 as output
    PORTD &= (0<<PD4);
    _delay_us(2);
    DDRD &= (0<<PD4);
    PORTD &= (0<<PD4);
    _delay_us(10);
    x = PIND;
    x &= 0x10;
    x = x >> 4;
    _delay_us(49);
    return x;
}

void one_wire_transmit_bit (unsigned int y)
{
    DDRD |= (1<<PD4); //set PD4 as output
    PORTD &= (0<<PD4);
    _delay_us(2);
    if (y == 1) PORTD |= (1<<PD4);
    _delay_us(58);
    DDRD &= (0<<PD4);
    PORTD &= (0<<PD4);
    _delay_us(1);   //recovery time
}

unsigned int one_wire_receive_byte (void)
{
    unsigned int value=0, x = 0;
    for (int i=0; i<8; i++)
    {
        value = value >> 1;
        x = one_wire_receive_bit();
        if (x == 1) value |= 0x80;
    }
    return value;
}

void one_wire_transmit_byte (unsigned int z)
{
    for (int i=0; i<8; i++)
    {
        one_wire_transmit_bit (z & 0x01);
        z = z >> 1;
    }
}

int _temperature_value (void)
{
    int value_H=0, value_L=0, x;
    x = one_wire_reset();
    if (x==0) return 0x8000;
    one_wire_transmit_byte (0xCC);
    one_wire_transmit_byte (0x44);
    x = 0;
    while (x == 0)
    {
        x = one_wire_receive_bit();
    }
    x = one_wire_reset();
    if (x==0) return 0x8000;
    one_wire_transmit_byte (0xCC);
    one_wire_transmit_byte (0xBE);
    value_L = one_wire_receive_byte();
    value_H = one_wire_receive_byte();
    asm("nop");
    int temp2 = value_H & 0xF8;
    if (temp2 == 0xF8) _sign = 1;   //negative value
    value_L &= 0xFF;        //isolate the 8 LSB
    value_H &= 0x07;        //discard sign bits
    value_H = value_H << 8; //shift bits 0-2 to position 8-10
    value_L |= value_H; //and combine the value in the 16bit variable
    return value_L;

}

//lab08 -> UART
//routines erwtima 8.1
void usart_init(unsigned int ubrr){
    UCSR0A=0;
    UCSR0B=(1<<RXEN0)|(1<<TXEN0);
    UBRR0H=(unsigned char)(ubrr>>8);
    UBRR0L=(unsigned char)ubrr;
    UCSR0C=(3 << UCSZ00);
    return;
}

void usart_transmit(uint8_t data){
    while(!(UCSR0A&(1<<UDRE0)));
    UDR0=data;
}
```

```
void usart_transmit_string(const char *msg){ //transmit entire string
    int i = 0;
    while(msg[i] != '\0'){
        usart_transmit(msg[i]);
        ++i;
    }
}

uint8_t usart_receive(){
    while(!(UCSR0A&(1<<RXC0)));
    return UDR0;
}

void usart_receive_string(char array[]) { //receive entire string, no
need to return array since input_arr is passed by reference
    uint8_t temp;
    int j = 0;
    while((temp=usart_receive())!='\n'){
                array[j]=temp;
                ++j;
           }
}

void success_or_fail(char array[],int time){
    DDRD |= 0b11111111;
    lcd_init();
    lcd_data(time + '0');
    _delay_ms(15);
    lcd_data('.');
    _delay_ms(15);
    if(array[0]=='"' && array[1]=='S') {
            lcd_data('S');
            _delay_ms(15);
                lcd_data('u');
            _delay_ms(15);
                lcd_data('c');
            _delay_ms(15);
                lcd_data('c');
            _delay_ms(15);
                lcd_data('e');
            _delay_ms(15);
                lcd_data('s');
            _delay_ms(15);
                lcd_data('s');
            _delay_ms(15);
    }
    if(array[0]=='"' && array[1]=='F') {
            lcd_data('F');
            _delay_ms(15);
                lcd_data('a');
            _delay_ms(15);
                lcd_data('i');
            _delay_ms(15);
                lcd_data('l');
            _delay_ms(15);
    }
    if(array[0]=='2' && array[1]=='0') {
            lcd_data('2');
            _delay_ms(15);
                lcd_data('0');
            _delay_ms(15);
                lcd_data('0');
            _delay_ms(15);
                lcd_data(' ');
            _delay_ms(15);
            lcd_data('O');
            _delay_ms(15);
            lcd_data('K');
            _delay_ms(15);
    }
    }

//routines erwtima 8.2
void check_pr(unsigned int a[]){
    DDRC |= 0b00000000;
    ADMUX |= 0b01000000; //ADC0, ADLAR=0 -> left adjusted
    ADCSRA |=0b11000111; // ADC enable + enable conversion
    unsigned int temp=0;
    while ((ADCSRA & (1 << ADSC)) != 0) //stuck here till conversion
ends (ADSC = 0)
        {

        }
    temp = ((ADC*2)/1024);
    a[0] = temp;
    temp  = (((ADC*2)%1024)*10)/1024;
    a[1] = temp;
    temp  = ((((ADC*2)%1024)*10)%1024)*10/1024;
    a[2] = temp;
}

int check_stat(unsigned int arr1[],unsigned int arr2[], unsigned char
key ){

    if (key == '7') return 1; //nurse call
    if(key == '#') {
        if(((arr1[0]>=1)&&(arr1[1]>=2)) ||
((arr1[0]<=0)&&(arr1[1]<=4))) return 2; //check pressure
        if(((arr2[0]>=3)&&(arr2[1]>=7)) ||
((arr2[0]<=3)&&(arr2[1]<=4))) return 3; //check temperature
    }
    if(((arr1[0]>=1)&&(arr1[1]>=2)) || ((arr1[0]<=0)&&(arr1[1]<=4)))
return 2; //check pressure either way
```

```
    if(((arr2[0]>=3)&&(arr2[1]>=7)) || ((arr2[0]<=3)&&(arr2[1]<=4)))
return 3; //check temperature either way
    return 0;
}


void lcd_ok(void) {
    lcd_data('O');
    _delay_ms(15);
    lcd_data('K');
    _delay_ms(15);
}

void lcd_nurse(void) {
    lcd_data('C');
    _delay_ms(15);
    lcd_data('A');
    _delay_ms(15);
    lcd_data('L');
    _delay_ms(15);
    lcd_data('L');
    _delay_ms(15);
    lcd_data(' ');
    _delay_ms(15);
    lcd_data('N');
    _delay_ms(15);
    lcd_data('U');
    _delay_ms(15);
    lcd_data('R');
    _delay_ms(15);
    lcd_data('S');
    _delay_ms(15);
    lcd_data('E');
    _delay_ms(15);
}

void lcd_pressure(void) {
    lcd_data('C');
    _delay_ms(15);
    lcd_data('H');
    _delay_ms(15);
    lcd_data('E');
    _delay_ms(15);
    lcd_data('C');
    _delay_ms(15);
    lcd_data('K');
    _delay_ms(15);
    lcd_data(' ');
    _delay_ms(15);
    lcd_data('P');
    _delay_ms(15);
    lcd_data('R');
    _delay_ms(15);
    lcd_data('E');
    _delay_ms(15);
    lcd_data('S');
    _delay_ms(15);
    lcd_data('S');
    _delay_ms(15);
}

void lcd_temp(void) {
    lcd_data('C');
    _delay_ms(15);
    lcd_data('H');
    _delay_ms(15);
    lcd_data('E');
    _delay_ms(15);
    lcd_data('C');
    _delay_ms(15);
    lcd_data('K');
    _delay_ms(15);
    lcd_data(' ');
    _delay_ms(15);
    lcd_data('T');
    _delay_ms(15);
    lcd_data('E');
    _delay_ms(15);
    lcd_data('M');
    _delay_ms(15);
    lcd_data('P');
    _delay_ms(15);
}

void lcd_print_values(unsigned int t_a[], unsigned int b[], int c){
    DDRD |= 0b11111111;
    lcd_init();
    lcd_data('T');
    _delay_ms(15);
    lcd_data(':');
    _delay_ms(15);
    lcd_data(t_a[0]+'0');
    _delay_ms(15);
    lcd_data(t_a[1]+'0');
    _delay_ms(15);
    lcd_data('.');
    _delay_ms(15);
    lcd_data(t_a[2]+'0');
    _delay_ms(15);
    lcd_data(' ');
    _delay_ms(15);
    lcd_data('P');
    _delay_ms(15);
    lcd_data(':');
    _delay_ms(15);
    lcd_data(b[0]+'0');
    _delay_ms(15);
    lcd_data(b[1]+'0');
    _delay_ms(15);
    lcd_data('.');
    _delay_ms(15);
    lcd_data(b[2]+'0');
    _delay_ms(15);
    lcd_command (0xC0);
    _delay_ms(80);
    lcd_data('S');
    _delay_ms(15);
    lcd_data('t');
    _delay_ms(15);
    lcd_data(c+'0');
    _delay_ms(15);
    lcd_data(':');
    _delay_ms(15);
    if(c==0) lcd_ok();
    if(c==1) lcd_nurse();
    if(c==2) lcd_pressure();
    if(c==3) lcd_temp();
}

void write_temp_arr(int a,unsigned int arr[]){
    int number1=0, number2=0, number3=0, check = 0;
    float dec=0;
    for (int i=4; i>0; i--)
    {
        check = a & 0x01;
        if (check == 1) dec += 1/(pow(2,i));
        a = a >> 1;
        check = 0;
    }
    a=a+14;
    number1 = a/100;
    number2 = (a-(number1*100))/10;
    arr[0]=number2;
    number3 = a-(number1*100 + number2*10);
    arr[1]=number3;
    if(dec!=0) {
        lcd_data('.');
        check = dec*10;
        arr[2]=check;
    }
    else {
        arr[2]=0;
    }
}

//routines erwtima 8.3
```

```
void usart_transmit_payload_all(unsigned int arr1[],unsigned int
arr2[], int stat) {
    usart_transmit_string("ESP:payload: [{\"name\":
\"temperature\",\"value\":\"");
    for (int i =0; i<3; ++i){
        if(i==0 && arr1[i]==0) continue;
        if(i==2) usart_transmit('.');
        usart_transmit(arr1[i]+'0');
    }
    usart_transmit_string("\"},{\"name\": \"pressure\",\"value\":\"");
    for (int i =0; i<3; ++i){
        if(i==0 && arr2[i]==0) continue;
        if(i==2) usart_transmit('.');
        usart_transmit(arr2[i]+'0');
    }
    usart_transmit_string("\"},{\"name\": \"team\",\"value\":\"17\"");
    if(stat==0) usart_transmit_string("},{\"name\":
\"status\",\"value\":\"OK\"");
    if(stat==1) usart_transmit_string("},{\"name\":
\"status\",\"value\":\"NURSECALL\"");
    if(stat==2) usart_transmit_string("},{\"name\":
\"status\",\"value\":\"CHECKPRESSURE\"");
    if(stat==3) usart_transmit_string("},{\"name\":
\"status\",\"value\":\"CHECKTEMP\"");
    usart_transmit_string("}]\n");
    }

void usart_lcd(char array[]) {
    lcd_init();
    lcd_data('4');
    _delay_ms(15); //minor delay
    lcd_data('.');
    _delay_ms(15); //minor delay
    int i=0;
    while((array[i])!='\0'){
        lcd_data(array[i]);
        _delay_ms(15); //minor delay
        ++i;
           }
    }

int main() {
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1's
bit4-7 as input and bit0-3 as output
    unsigned int temp2, pr[3], tp[3], stat;
    int n = 10; //size of receive array
    int nurse_flag=0;
    char arr1[n], arr2[n], arr3[n], arr4[n];
    unsigned char button;
    DDRD |= 0b11111111;  // output for LCD
    usart_transmit_string("ESP:restart\n");
    while(1){ //to repeat entire procedure
        int counter = 0;
        DDRD |= 0b11111111;  // output for LCD

        //erwtima 8.1
        memset(arr1,'\0',n);
        memset(arr2,'\0',n);
        usart_init(103); //anti gia UBRR0
        usart_transmit_string("ESP:connect\n"); //first cmd
        _delay_ms(100);
        usart_receive_string(arr1);
        _delay_ms(100);
        success_or_fail(arr1,1);
        _delay_ms(350); //minor delay

usart_transmit_string("ESP:url:\"http://192.168.1.250:5000/data\"\n");
//second cmd
        _delay_ms(100);
        usart_receive_string(arr2);
        _delay_ms(100);
        success_or_fail(arr2,2);
        _delay_ms(350); //minor delay

        //erwtima 8.2
        button = keypad_to_ascii();
        asm("NOP");
        for(int i = 0; i <100; ++i){
            ++counter;
            button = keypad_to_ascii();
            _delay_ms(30);
            if(button=='7'){
                nurse_flag =1;
                break;
            }
            if (button == '#'){
                 nurse_flag =0;
                break;
            }
        }
        check_pr(pr);
        temp2 = _temperature_value();
        write_temp_arr(temp2,tp);
        stat = check_stat(pr, tp, button);
        if (nurse_flag == 1) stat = 1;
        lcd_print_values(tp, pr, stat);
        _delay_ms(500);

        //erwtima 8.3
        memset(arr3,'\0',n);
        memset(arr4,'\0',n); //array of nullbytes
        usart_transmit_payload_all(tp,pr,stat);
        _delay_ms(100);
        usart_receive_string(arr3);
        _delay_ms(100);
        success_or_fail(arr3,3);
        _delay_ms(350); //minor delay
        usart_transmit_string("ESP:transmit\n");
        _delay_ms(100);
        usart_receive_string(arr4);
        _delay_ms(100);
        success_or_fail(arr4,4);
        _delay_ms(350);
    }
    return 0;
}
```